



Vulkan[®] SC 1.0.20 - A Specification
(with all registered Vulkan SC
extensions)

Based on Vulkan 1.2.330

The Khronos[®] Vulkan SC Working Group

Version 1.0.20, 2025-11-19 21:18:25Z: from git branch: github-sc_main commit:
df93b82d4036924cdf735fa83077c7d10d65d7d9

Table of Contents

1. Preamble	1
2. Introduction	3
2.1. Safety Critical Philosophy	3
2.2. Document Conventions	4
3. Fundamentals	8
3.1. Host and Device Environment	8
3.2. Execution Model	8
3.3. Object Model	10
3.4. Application Binary Interface	12
3.5. Command Syntax and Duration	13
3.6. Threading Behavior	17
3.7. Valid Usage	26
3.8. VkResult Return Codes	33
3.9. Numeric Representation and Computation	36
3.10. Fixed-Point Data Conversions	38
3.11. String Representation	40
3.12. Common Object Types	40
3.13. API Name Aliases	57
4. Initialization	58
4.1. Command Function Pointers	58
4.2. Instances	61
5. Devices and Queues	75
5.1. Physical Devices	75
5.2. Devices	108
5.3. Queues	129
6. Command Buffers	138
6.1. Command Buffer Lifecycle	138
6.2. Command Pools	140
6.3. Command Buffer Allocation and Management	148
6.4. Command Buffer Recording	153
6.5. Command Buffer Submission	160
6.6. Queue Forward Progress	179
6.7. Secondary Command Buffer Execution	179
6.8. Command Buffer Device Mask	183
7. Synchronization and Cache Control	187
7.1. Execution and Memory Dependencies	187
7.2. Implicit Synchronization Guarantees	213
7.3. Fences	215

7.4. Semaphores	245
7.5. Events	277
7.6. Pipeline Barriers	301
7.7. Memory Barriers	310
7.8. Wait Idle Operations	345
7.9. Host Write Ordering Guarantees	347
7.10. Synchronization and Multiple Physical Devices	347
7.11. Calibrated Timestamps	347
8. Render Pass	352
8.1. Render Pass Creation	353
8.2. Render Pass Compatibility	407
8.3. Framebuffers	408
8.4. Render Pass Load Operations	419
8.5. Render Pass Store Operations	420
8.6. Render Pass Multisample Resolve Operations	421
8.7. Render Pass Commands	423
8.8. Common Render Pass Data Races (Informative)	446
9. Shaders	448
9.1. Shader Modules	448
9.2. Binding Shaders	450
9.3. Shader Execution	450
9.4. Shader Out-of-Bounds Memory Access	451
9.5. Shader Memory Access Ordering	455
9.6. Shader Inputs and Outputs	455
9.7. Vertex Shaders	456
9.8. Tessellation Control Shaders	456
9.9. Tessellation Evaluation Shaders	458
9.10. Geometry Shaders	458
9.11. Fragment Shaders	459
9.12. Compute Shaders	459
9.13. Interpolation Decorations	459
9.14. Static Use	460
9.15. Scope	460
9.16. Group Operations	464
9.17. Quad Group Operations	466
9.18. Derivative Operations	466
9.19. Helper Invocations	468
10. Pipelines	469
10.1. Multiple Pipeline Creation	470
10.2. Pipeline Creation Flags	471
10.3. Compute Pipelines	472

10.4. Graphics Pipelines	482
10.5. Pipeline Destruction	505
10.6. Pipeline Derivatives	506
10.7. Pipeline Cache	506
10.8. Offline Pipeline Compilation	516
10.9. Pipeline Memory Reservation	517
10.10. Pipeline Identifier	517
10.11. Specialization Constants	519
10.12. Pipeline Binding	523
10.13. Dynamic State	525
11. Memory Allocation	527
11.1. Host Memory	527
11.2. Device Memory	527
12. Resource Creation	582
12.1. Buffers	582
12.2. Buffer Views	590
12.3. Buffer Device Addresses	595
12.4. Images	597
12.5. Image Layouts	628
12.6. Image Views	633
12.7. Resource Memory Association	651
12.8. Resource Sharing Mode	679
12.9. Memory Aliasing	682
13. Images	685
13.1. Image Coordinate Validation	686
13.2. Image Reads	686
13.3. Image Writes	687
14. Samplers	688
14.1. Sampler $Y'C_B C_R$ Conversion	699
15. Resource Descriptors	709
15.1. Descriptor Types	709
15.2. Descriptor Sets	713
15.3. Physical Storage Buffer Access	770
16. Shader Interfaces	771
16.1. Shader Input and Output Interfaces	771
16.2. Vertex Input Interface	774
16.3. Fragment Output Interface	774
16.4. Fragment Input Attachment Interface	775
16.5. Shader Resource Interface	776
16.6. Built-In Variables	784
17. Image Operations	811

17.1. Image Operations Overview	811
17.2. Sampling Operations	814
17.3. Normalized Texel Coordinate Operations	830
17.4. Unnormalized Texel Coordinate Operations	836
17.5. Integer Texel Coordinate Operations	838
17.6. Image Sample Operations	838
17.7. Image Operation Steps	843
17.8. Image Query Instructions	843
18. Queries	845
18.1. Query Pools	845
18.2. Query Operation	851
18.3. Occlusion Queries	868
18.4. Pipeline Statistics Queries	869
18.5. Timestamp Queries	871
18.6. Performance Queries	877
19. Clear Commands	882
19.1. Clearing Images Outside a Render Pass Instance	882
19.2. Clearing Images Inside a Render Pass Instance	888
19.3. Clear Values	891
19.4. Filling Buffers	893
19.5. Updating Buffers	895
20. Copy Commands	899
20.1. Copying Data Between Buffers	899
20.2. Copying Data Between Images	905
20.3. Copying Data Between Buffers and Images	923
20.4. Image Copies With Scaling	948
20.5. Resolving Multisample Images	962
20.6. Object Refreshes	974
21. Drawing Commands	978
21.1. Primitive Topologies	980
21.2. Effective Primitive Topology	989
21.3. Primitive Order	990
21.4. Programmable Primitive Shading	990
22. Fixed-Function Vertex Processing	1075
22.1. Vertex Attributes	1075
22.2. Vertex Input Description	1079
22.3. Vertex Attribute Divisor in Instanced Rendering	1092
22.4. Vertex Input Address Calculation	1094
23. Tessellation	1096
23.1. Tessellator	1096
23.2. Tessellator Patch Discard	1098

23.3. Tessellator Spacing	1099
23.4. Tessellation Primitive Ordering	1099
23.5. Tessellator Vertex Winding Order	1100
23.6. Triangle Tessellation	1100
23.7. Quad Tessellation	1102
23.8. Isoline Tessellation	1104
23.9. Tessellation Point Mode	1104
23.10. Tessellation Pipeline State	1104
24. Geometry Shading	1107
24.1. Geometry Shader Input Primitives	1107
24.2. Geometry Shader Output Primitives	1108
24.3. Multiple Invocations of Geometry Shaders	1108
24.4. Geometry Shader Primitive Ordering	1108
25. Fixed-Function Vertex Post-Processing	1109
25.1. Flat Shading	1109
25.2. Primitive Clipping	1109
25.3. Clipping Shader Outputs	1111
25.4. Coordinate Transformations	1112
25.5. Controlling the Viewport	1113
26. Rasterization	1122
26.1. Discarding Primitives Before Rasterization	1126
26.2. Rasterization Order	1128
26.3. Multisampling	1128
26.4. Custom Sample Locations	1133
26.5. Fragment Shading Rates	1136
26.6. Sample Shading	1145
26.7. Points	1145
26.8. Line Segments	1146
26.9. Polygons	1156
27. Fragment Operations	1170
27.1. Discard Rectangles Test	1171
27.2. Scissor Test	1177
27.3. Sample Mask Test	1179
27.4. Fragment Shading	1179
27.5. Multisample Coverage	1181
27.6. Depth and Stencil Operations	1182
27.7. Depth Bounds Test	1184
27.8. Stencil Test	1187
27.9. Depth Test	1196
27.10. Sample Counting	1200
27.11. Coverage Reduction	1200

28. The Framebuffer	1202
28.1. Blending	1202
28.2. Logical Operations	1221
28.3. Color Write Mask	1225
28.4. Color Write Enable	1225
29. Dispatching Commands	1229
30. Sparse Resources	1246
30.1. Sparse Resource Features	1246
30.2. Sparse Resource API	1248
31. Window System Integration (WSI)	1250
31.1. WSI Platform	1250
31.2. WSI Surface	1250
31.3. Presenting Directly to Display Devices	1252
31.4. Querying for WSI Support	1281
31.5. Surface Queries	1283
31.6. Device Group Queries	1305
31.7. WSI Swapchain	1310
31.8. HDR Metadata	1346
32. Extending Vulkan	1349
32.1. Functionality Levels	1349
32.2. Instance and Device Versions	1349
32.3. Core Versions	1349
32.4. Layers	1352
32.5. Extensions	1355
32.6. Extension Dependencies	1359
32.7. Compatibility Guarantees (Informative)	1360
33. Features	1365
33.1. Feature Requirements	1432
34. Limits	1436
34.1. Limit Requirements	1473
34.2. Additional Multisampling Capabilities	1492
35. Formats	1494
35.1. Format Definition	1494
35.2. Format Properties	1542
35.3. Required Format Support	1551
36. Additional Capabilities	1572
36.1. Additional Image Capabilities	1572
36.2. Additional Buffer Capabilities	1590
36.3. Optional Semaphore Capabilities	1592
36.4. Optional Fence Capabilities	1597
36.5. Timestamp Calibration Capabilities	1602

36.6. Object Refresh Capabilities	1603
37. Debugging	1605
37.1. Debug Utilities	1608
37.2. Fault Handling	1629
Appendix A: Vulkan Environment for SPIR-V	1637
Versions and Formats	1637
Capabilities	1637
Validation Rules Within a Module	1644
Precision and Operation of SPIR-V Instructions	1667
Buffer Indexing Calculations	1673
Signedness of SPIR-V Image Accesses	1674
Image Format and Type Matching	1674
Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types	1676
Compatibility Between SPIR-V Image Formats and Vulkan Formats	1677
Appendix B: Memory Model	1679
Agent	1679
Memory Location	1679
Allocation	1679
Memory Operation	1680
Reference	1680
Program-Order	1680
Scope	1681
Atomic Operation	1681
Scoped Modification Order	1682
Memory Semantics	1682
Release Sequence	1684
Synchronizes-With	1684
System-Synchronizes-With	1686
Private vs. Non-Private	1686
Inter-Thread-Happens-Before	1686
Happens-Before	1687
Availability and Visibility	1687
Availability, Visibility, and Domain Operations	1689
Availability and Visibility Semantics	1690
Per-Instruction Availability and Visibility Semantics	1690
Location-Ordered	1691
Data Race	1692
Visible-To	1692
Acyclicity	1692
Shader I/O	1693
Deallocation	1694

Descriptions (Informative)	1694
Tessellation Output Ordering	1694
Appendix C: Compressed Image Formats	1696
Block-Compressed Image Formats	1697
ETC Compressed Image Formats	1698
ASTC Compressed Image Formats	1699
Appendix D: Core Revisions (Informative)	1701
Vulkan Version 1.2	1701
Vulkan Version 1.1	1709
Vulkan Version 1.0	1720
Vulkan SC Version 1.0	1733
Appendix E: Layers & Extensions (Informative)	1736
Extension Dependencies	1736
Extension Interactions	1736
List of Current Extensions	1736
List of Deprecated Extensions	1930
Appendix F: Legacy and Superseded Functionality	1940
List of Legacy Functionality	1940
Appendix G: API Boilerplate	1942
Vulkan Header Files	1942
Window System-Specific Header Control (Informative)	1946
Provisional Extension Header Control (Informative)	1948
Appendix H: Invariance	1950
Repeatability	1950
Multi-Pass Algorithms	1950
Invariance Rules	1950
Tessellation Invariance	1952
Appendix I: Vulkan SC Deviations From Base Vulkan	1954
Additions	1954
Modifications	1955
Removals	1959
Extension Support	1963
Fault and Error Handling	1963
Undefined Behavior in the API	1964
MISRA C:2012 Deviations	1964
Appendix J: Lexicon	1967
Glossary	1967
Common Abbreviations	1990
Prefixes	1991
Appendix K: Credits (Informative)	1993
Working Group Contributors to Vulkan SC 1.0	1993

Working Group Contributors to Vulkan.....	1995
Other Credits.....	2007

Chapter 1. Preamble

Copyright 2014-2025 The Khronos Group Inc.

This Specification is protected by copyright laws and contains material proprietary to Khronos. Except as described by these terms, it or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Khronos.

Khronos grants a conditional copyright license to use and reproduce the unmodified Specification for any purpose, without fee or royalty, EXCEPT no licenses to any patent, trademark or other intellectual property rights are granted under these terms.

Khronos makes no, and expressly disclaims any, representations or warranties, express or implied, regarding this Specification, including, without limitation: merchantability, fitness for a particular purpose, non-infringement of any intellectual property, correctness, accuracy, completeness, timeliness, and reliability. Under no circumstances will Khronos, or any of its Promoters, Contributors or Members, or their respective partners, officers, directors, employees, agents or representatives be liable for any damages, whether direct, indirect, special or consequential damages for lost revenues, lost profits, or otherwise, arising from or in connection with these materials.

This document contains extensions which are not ratified by Khronos, and as such is not a ratified Specification, though it contains text from (and is a superset of) the ratified Specification.

The ratified version of the Vulkan SC Specification can be found at <https://registry.khronos.org/vulkansc/specs/latest-ratified/pdf/vkspec.pdf> (core with all ratified extensions).

This Specification contains substantially unmodified functionality from, and is a successor to, Khronos specifications including Vulkan, OpenGL SC OpenGL, OpenGL ES and OpenCL.

The Khronos Intellectual Property Rights Policy defines the terms 'Scope', 'Compliant Portion', and 'Necessary Patent Claims'.

Some parts of this Specification are purely informative and so are EXCLUDED the Scope of this Specification. The [Document Conventions](#) section of the [Introduction](#) defines how these parts of the Specification are identified.

Where this Specification uses [technical terminology](#), defined in the [Glossary](#) or otherwise, that refer to enabling technologies that are not expressly set forth in this Specification, those enabling technologies are EXCLUDED from the Scope of this Specification. For clarity, enabling technologies not disclosed with particularity in this Specification (e.g. semiconductor manufacturing technology, hardware architecture, processor architecture or microarchitecture, memory architecture, compiler technology, object oriented technology, basic operating system technology, compression technology, algorithms, and so on) are NOT to be considered expressly set forth; only those application program interfaces and data structures disclosed with particularity are included in the Scope of this Specification.

For purposes of the Khronos Intellectual Property Rights Policy as it relates to the definition of

Necessary Patent Claims, all recommended or optional features, behaviors and functionality set forth in this Specification, if implemented, are considered to be included as Compliant Portions.

Where this Specification identifies specific sections of external references, only those specifically identified sections define [normative](#) functionality. The Khronos Intellectual Property Rights Policy excludes external references to materials and associated enabling technology not created by Khronos from the Scope of this Specification, and any licenses that may be required to implement such referenced materials and associated technologies must be obtained separately and may involve royalty payments.

Khronos and Vulkan are registered trademarks, and SPIR-V is a trademark of The Khronos Group Inc. OpenCL is a trademark of Apple Inc., used under license by Khronos. OpenGL is a registered trademark and the OpenGL ES logo is a trademark of Hewlett Packard Enterprise, used under license by Khronos. ASTC is a trademark of ARM Holdings PLC. All other product names, trademarks, and/or company names are used solely for identification and belong to their respective owners.

Chapter 2. Introduction

This document, referred to as the “Vulkan SC Specification”, “Vulkan Specification” or just the “Specification” hereafter, describes the Vulkan SC Application Programming Interface (API). “Base Vulkan Specification” refers to the Vulkan Specification (<https://registry.khronos.org/vulkan/>) that Vulkan SC is based on. “Vulkan” and “Vulkan SC” refer to the Vulkan SC API and “Base Vulkan” refers to the Vulkan API that Vulkan SC is based on. Vulkan is a C99 API designed for explicit control of low-level graphics and compute functionality.

The canonical version of the Specification is available in the official [Vulkan SC Registry](https://registry.khronos.org/vulkansc/) (<https://registry.khronos.org/vulkansc/>). The source files used to generate the Vulkan SC specification are stored in the [Vulkan SC Documentation Repository](https://github.com/KhronosGroup/VulkanSC-Docs) (<https://github.com/KhronosGroup/VulkanSC-Docs>). The source repository additionally has a public issue tracker and allows the submission of pull requests that improve the specification.

2.1. Safety Critical Philosophy

Vulkan SC 1.0.20 is based on Vulkan 1.2 and, except where explicitly noted, supports all of the same features, properties, and limits as Vulkan 1.2.

Throughout the Vulkan SC specification, changes have been made to the Base Vulkan Specification in order to align it with safety critical use cases and certification. In general changes were made to meet the following categories:

- Deterministic Execution (predictable execution times and results)
- Robustness (error handling, removing ambiguity, clarifying undefined behavior)
- Simplification (changes made to reduce certification effort and challenges)

To simplify capturing the reasoning behind deviations made from the Base Vulkan Specification, the Vulkan SC specification utilizes change identifications to give the reader insight into why the change was made in a concise manner. The change identifications are captured in [Change Justification Table](#). In addition, the Vulkan SC specification contains [Vulkan SC Deviations From Base Vulkan](#) which is a complete list of changes between Base Vulkan and Vulkan SC. This is targeted at readers who are familiar with Base Vulkan and would like to understand the differences between Vulkan SC and the Base Vulkan specification. Vulkan SC was designed to maintain as much compatibility as possible with base Vulkan. Some functionality has been removed, some new functionality has been added, but where the APIs overlap the only changes are to Valid Usage rules. All functions and structures that exist in both APIs have the same signatures, and enums and defined constants that exist in both APIs have the same values.

Vulkan SC follows the Base Vulkan philosophy of requiring valid usage from the application. It is left to each implementation to determine how to ensure safe operation with respect to invalid usage. This **may** involve determining that certain invalid usage does not pose a safety risk, adding valid usage checks in the driver, requiring valid usage checks in the application, or some combination of these. Additionally, validation layers are supported during development.

2.1.1. Change Justification Table

The following is a list of the safety critical change identifications used to concisely capture the justification for deviations from the Base Vulkan Specification.

Table 1. Change Justifications

Change ID	Description
SCID-1	Deterministic behavior - no randomness or unpredictability, always produce the same output from a given starting condition or initial state
SCID-2	Asynchronous calls - calls initiated by the application but may not execute or use their parameter data until a later time shall be clearly defined when any parameter data is used, especially data which is passed by reference or pointer
SCID-3	Notification of change of state - avoid the use of asynchronous events causing code to execute (i.e. callbacks) as this can cause the worst case execution time of a system to be indeterminate
SCID-4	Garbage collection methods - avoid the use of garbage collection as this can cause the worst case execution time of a system to be indeterminate. Avoid memory fragmentation by deleting entire buffers instead of individual items within a buffer
SCID-5	Fully testable - all behavior of the API must be testable in a repeatable manner, consistent from test run to test run (in some cases this may mean testable by inspection)
SCID-6	Undefined behavior - the API must behave as expected under valid input conditions, clearly document conditions that would result in 'fatal error' leaving the system in an unrecoverable state, and document conditions that would result in undefined behavior based on invalid input
SCID-7	Unique ID - provide a facility to return a runtime implementation unique identifier specific to that runtime so that it may be interrogated at any time. For example, such information could be the version number, name, date, release build number or a combination of these that is unique and comprehensible
SCID-8	Code complexity - reducing code complexity to help facilitate certification (for example if there are multiple ways to do the same thing, potentially eliminating one or more of the alternative methods)

2.2. Document Conventions

The Vulkan specification is intended for use by both implementors of the API and application developers seeking to make use of the API, forming a contract between these parties. Specification text may address either party; typically the intended audience can be inferred from context, though some sections are defined to address only one of these parties. (For example, [Valid Usage](#) sections only address application developers). Any requirements, prohibitions, recommendations or options defined in specification text are imposed only on the audience of that text.

2.2.1. Normative Requirements

The Vulkan Specification uses a combination of [normative terminology](#) and [normative descriptions](#) to express the requirements that it imposes on applications and implementations. An application which complies with all normative requirements imposed on applications is said to make **valid use** of the API; failing to comply with such requirements results in undefined behavior, as discussed in the [Valid Usage](#) section below. In the context of this document, an implementation which complies with all normative requirements imposed on implementations is said to be **conformant**.

NOTE

The Khronos Group imposes additional requirements on implementors who wish to make public statements describing their Vulkan implementations as conformant. These include signing the Vulkan Adopter's Agreement, paying the associated fee, and making a successful conformance test submission to the Khronos Conformance Process. For details see the [Khronos Trademark Guidelines](https://www.khronos.org/legal/khronos-trademark-guidelines) (<https://www.khronos.org/legal/khronos-trademark-guidelines>).

Normative Terminology

Within this specification, the key words **must**, **required**, **should**, **may**, and **optional** are to be interpreted as described in [RFC 2119 - Key words for use in RFCs to Indicate Requirement Levels](https://www.ietf.org/rfc/rfc2119.txt) (<https://www.ietf.org/rfc/rfc2119.txt>). The additional key word **optionally** is an alternate form of **optional**, for use where grammatically appropriate. These key words are highlighted in the specification to indicate that they are being used in a specific technical sense.

The additional key words **can** and **cannot** are to be interpreted as describing the capabilities of an application, as follows:

can

This word means that the application is able to perform the action described.

cannot

This word means that the API and/or the execution environment provide no mechanism through which the application can express or accomplish the action described.

These key words are never used in text addressing implementors.

NOTE

There is an important distinction between **cannot** and **must not**, as used in this Specification. **Cannot** refers to something the API provides no way for the application to express or accomplish. **Must not** describes something that the application is able to express, but that is not valid use of the API, and will have undefined and potentially unrecoverable consequences.

Normative Descriptions

In the Vulkan Specification, the normative term **must** is primarily used to describe **application** behavior, and in particular to constrain what inputs or commands issued by the application to the implementation are considered valid.

To constrain **implementation** behavior, the specification sometimes uses **must**, but more often

simply describes the behavior of the implementation in response to specified commands and inputs. Unless explicitly stated otherwise, such references to implementation behavior describe the behavior of **conformant** implementations, and express normative requirements which an implementation must satisfy in order to conform to the specification. For example, if the specification says “Under **specified condition**, the error code `VK_ERROR_FEATURE_NOT_PRESENT` is returned”, that behavior is a requirement of the specification, and an implementation which does not return that error code under that condition is not conformant.

When the normative terms **may**, **should**, or **optional** are used to describe implementation behavior, they define alternative or optional behaviors which a conformant implementation may or may not exhibit. Such statements are also normative. For example, if the specification says “Under **specified condition**, the implementation **should** return A but **may** instead return B”, then an implementation that returns either A or B under that condition is conformant (assuming it does not violate other normative requirements), while an implementation that returns anything else is not.

2.2.2. Normative References

References to external documents are considered normative references if the Specification uses [normative terminology](#) or [normative descriptions](#) to refer to them or their requirements, either as a whole or in part.

The following documents are referenced by normative sections of the specification:

IEEE. August, 2008. *IEEE Standard for Floating-Point Arithmetic*. IEEE Std 754-2008. <https://dx.doi.org/10.1109/IEEESTD.2008.4610935>.

Andrew Garrard. *Khronos Data Format Specification, version 1.3*. <https://registry.khronos.org/DataFormat/specs/1.3/dataformat.1.3.html>.

John Kessenich. *SPIR-V Extended Instructions for GLSL, Version 1.00* (February 10, 2016). <https://registry.khronos.org/spir-v/>.

John Kessenich, Boaz Ouriel, and Raun Krisch. *SPIR-V Specification, Version 1.5, Revision 3, Unified* (April 24, 2020). <https://registry.khronos.org/spir-v/>.

ITU-T. *H.264 Advanced Video Coding for Generic Audiovisual Services* (August, 2021). <https://www.itu.int/rec/T-REC-H.264-202108-I/>.

ITU-T. *H.265 High Efficiency Video Coding* (August, 2021). <https://www.itu.int/rec/T-REC-H.265-202108-S/>.

Google. *VP9 Bitstream & Decoding Process Specification* (February 22, 2017). <https://storage.googleapis.com/downloads.webmproject.org/docs/vp9/vp9-bitstream-specification-v0.7-20170222-draft.pdf>.

Alliance for Open Media. *AV1 Bitstream & Decoding Process Specification* (January 8, 2019). <https://aomediacodec.github.io/av1-spec/av1-spec.pdf>.

Jon Leech. *The Khronos Vulkan API Registry* (February 26, 2023). <https://registry.khronos.org/>

vulkan/specs/latest/registry.html .

Jon Leech and Tobias Hector. *Vulkan Documentation and Extensions: Procedures and Conventions* (February 26, 2023). <https://registry.khronos.org/vulkan/specs/latest/styleguide.html> .

Architecture of the Vulkan Loader Interfaces (October, 2021). <https://github.com/KhronosGroup/Vulkan-Loader/blob/main/docs/LoaderInterfaceArchitecture.md> .

2.2.3. Informative Language

Some language in the specification is purely informative, intended to provide background information or make suggestions to implementors or developers. Such language does not impose normative requirements on implementations or applications.

All NOTES are implicitly informative.

If an entire chapter, section, or appendix contains only informative language, its title will be suffixed with “(Informative)”. Unless so noted in the title, all chapters, sections, and appendices in this document are normative.

2.2.4. Technical Terminology

The Vulkan Specification makes use of common engineering and graphics terms such as **Pipeline**, **Shader**, and **Host** to identify and describe Vulkan API constructs and their attributes, states, and behaviors. The [Glossary](#) defines the basic meanings of these terms in the context of the Specification. The Specification text provides fuller definitions of the terms and may elaborate, extend, or clarify the [Glossary](#) definitions. When a term defined in the [Glossary](#) is used in normative language within the Specification, the definitions within the Specification govern and supersede any meanings the terms may have in other technical contexts (i.e. outside the Specification).

2.2.5. Ratification

Ratification of a Vulkan core version or extension is a status conferred by vote of the Khronos Board of Promoters, bringing that core version or extension under the umbrella of the Khronos IP Rights Policy.

All Vulkan core versions and **KHR** extensions (including provisional specifications) are ratified, as are some multi-vendor **EXT** extensions. Ratification status of extensions is described in the [Layers & Extensions \(Informative\)](#) appendix.

NOTE

Ratification status is primarily of interest to IHVs developing GPU hardware and Vulkan implementations. For developers, ratification does not necessarily mean that an extension is “better”, has a more stable API, or is more widely supported than alternative ways of achieving that functionality.

Interactions between ratified and non-ratified extensions are not themselves ratified.

Chapter 3. Fundamentals

This chapter introduces fundamental concepts including the Vulkan architecture and execution model, API syntax, queues, pipeline configurations, numeric representation, state and state queries, and the different types of objects and shaders. It provides a framework for interpreting more specific descriptions of commands and behavior in the remainder of the Specification.

3.1. Host and Device Environment

The Vulkan Specification assumes and requires: the following properties of the host environment with respect to Vulkan implementations:

- The host **must** have runtime support for 8, 16, 32 and 64-bit signed and unsigned twos-complement integers, all addressable at the granularity of their size in bytes.
- The host **must** have runtime support for 32- and 64-bit floating-point types satisfying the range and precision constraints in the [Floating-Point Computation](#) section.
- The representation and endianness of these types on the host **must** match the representation and endianness of the same types on every physical device supported.

NOTE

Since a variety of data types and structures in Vulkan **may** be accessible by both host and physical device operations, the implementation **should** be able to access such data efficiently in both paths in order to facilitate writing portable and performant applications.

3.2. Execution Model

This section outlines the execution model of a Vulkan system.

Vulkan exposes one or more *devices*, each of which exposes one or more *queues* which **may** process work asynchronously to one another. The set of queues supported by a device is partitioned into *families*. Each family supports one or more types of functionality and **may** contain multiple queues with similar characteristics. Queues within a single family are considered *compatible* with one another, and work produced for a family of queues **can** be executed on any queue within that family. This specification defines the following types of functionality that queues **may** support: graphics, compute, protected memory management, and transfer.

NOTE

A single device **may** report multiple similar queue families rather than, or as well as, reporting multiple members of one or more of those families. This indicates that while members of those families have similar capabilities, they are *not* directly compatible with one another.

Device memory is explicitly managed by the application. Each device **may** advertise one or more heaps, representing different areas of memory. Memory heaps are either device-local or host-local, but are always visible to the device. Further detail about memory heaps is exposed via memory types available on that heap. Examples of memory areas that **may** be available on an implementation include:

- *device-local* is memory that is physically connected to the device.
- *device-local, host visible* is device-local memory that is visible to the host.
- *host-local, host visible* is memory that is local to the host and visible to the device and host.

On other architectures, there **may** only be a single heap that **can** be used for any purpose.

3.2.1. Queue Operation

Vulkan queues provide an interface to the execution engines of a device. Commands for these execution engines are recorded into command buffers ahead of execution time, and then submitted to a queue for execution. Once submitted to a queue, command buffers will begin and complete execution without further application intervention, though the order of this execution is dependent on a number of [implicit and explicit ordering constraints](#).

Work is submitted to queues using *queue submission commands* that typically take the form `vkQueue*` (e.g. `vkQueueSubmit`), and **can** take a list of semaphores upon which to wait before work begins and a list of semaphores to signal once work has completed. The work itself, as well as signaling and waiting on the semaphores are all *queue operations*. Queue submission commands return control to the application once queue operations have been submitted - they do not wait for completion.

There are no implicit ordering constraints between queue operations on different queues, or between queues and the host, so these **may** operate in any order with respect to each other. Explicit ordering constraints between different queues or with the host **can** be expressed with [semaphores](#) and [fences](#).

Command buffer submissions to a single queue respect [submission order](#) and other [implicit ordering guarantees](#), but otherwise **may** overlap or execute out of order. Other types of batches and queue submissions against a single queue have no implicit ordering constraints with any other queue submission or batch. Additional explicit ordering constraints between queue submissions and individual batches can be expressed with [semaphores](#) and [fences](#).

Before a fence or semaphore is signaled, it is guaranteed that any previously submitted queue operations have completed execution, and that memory writes from those queue operations are [available](#) to future queue operations. Waiting on a signaled semaphore or fence guarantees that previous writes that are available are also [visible](#) to subsequent commands.

Command buffer boundaries, both between primary command buffers of the same or different batches or submissions as well as between primary and secondary command buffers, do not introduce any additional ordering constraints. In other words, submitting the set of command buffers (which **can** include executing secondary command buffers) between any semaphore or fence operations execute the recorded commands as if they had all been recorded into a single primary command buffer, except that the current state is [reset](#) on each boundary. Explicit ordering constraints **can** be expressed with [explicit synchronization primitives](#).

There are a few [implicit ordering guarantees](#) between commands within a command buffer, but only covering a subset of execution. Additional explicit ordering constraints can be expressed with the various [explicit synchronization primitives](#).

NOTE

Implementations have significant freedom to overlap execution of work submitted to a queue, and this is common due to deep pipelining and parallelism in Vulkan devices.

Commands recorded in command buffers can perform actions, set state that persists across commands, synchronize other commands, or indirectly launch other commands, with some commands fulfilling several of these roles. The “Command Properties” section for each such command lists which of these roles the command takes:

Action

Action commands perform operations that can update values in memory. E.g. [draw commands](#), [dispatch commands](#).

State

State setting commands update the current state of a command buffer, affecting the operation of future action commands.

Synchronization

Synchronization commands impose ordering constraints on action commands, by introducing explicit [execution and memory dependencies](#).

Indirection

Indirection commands execute other commands which were not directly recorded in the same command buffer.

NOTE

In the absence of explicit synchronization or [implicit ordering guarantees](#), action commands may overlap execution or execute out of order, potentially leading to data races. However, such reordering does not affect the current state observed by any action command. Each action command uses the state in effect at the point where the command occurs in the command buffer, regardless of when it is executed.

3.3. Object Model

The devices, queues, and other entities in Vulkan are represented by Vulkan objects. At the API level, all objects are referred to by handles. There are two classes of handles, dispatchable and non-dispatchable. *Dispatchable* handle types are a pointer to an opaque type. This pointer **may** be used by layers as part of intercepting API commands, and thus each API command takes a dispatchable type as its first parameter. Each object of a dispatchable type **must** have a unique handle value during its lifetime.

Non-dispatchable handle types are a 64-bit integer type whose meaning is implementation-dependent. Non-dispatchable handles **may** encode object information directly in the handle rather than acting as a reference to an underlying object, and thus **may** not have unique handle values. If handle values are not unique, then destroying one such handle **must** not cause identical handles of other types to become invalid, and **must** not cause identical handles of the same type to become invalid if that handle value has been created more times than it has been destroyed.

All objects created or allocated from a **VkDevice** (i.e. with a **VkDevice** as the first parameter) are private to that device, and **must** not be used on other devices.

3.3.1. Object Lifetime

Objects are created or allocated by **vkCreate*** and **vkAllocate*** commands, respectively. Once an object is created or allocated, its “structure” is considered to be immutable, though the content of certain object types is still free to change. When an object is passed to another command, it **may** be accessed by the implementation, which **may** include both read and write access unless explicitly stated otherwise. Objects are destroyed or freed by **vkDestroy*** and **vkFree*** commands, respectively.

Objects that are allocated (rather than created) take resources from an existing pool object or memory heap, and when freed return resources to that pool or heap. While object creation and destruction are generally expected to be low-frequency occurrences during runtime, allocating and freeing objects **can** occur at high frequency. Pool objects help accommodate improved performance of the allocations and frees.

In Vulkan SC, data structures for objects are reserved by the implementation at device creation time in order to enable implementations to rely solely on static memory management at runtime. The **VkDeviceObjectReservationCreateInfo** structure provides upper bounds on the simultaneous number of objects of each type that **can** be allocated during the lifetime of the **VkDevice**. Most objects can be created and destroyed as needed, provided that no more than the requested number are in existence at any point in time.

Applications are responsible for managing the lifetimes of Vulkan objects and memory passed into the Vulkan API. The access semantics of different functions in the API follow a typical pattern as laid out below, with any exceptions listed with the commands or objects that have them.

Application-owned memory and Vulkan objects **may** be accessed at any time during the execution of a command they are passed to. Vulkan objects that device addresses are retrieved from **may** be accessed by the implementation any time that memory backing the device address is accessed. Device addresses and Vulkan objects passed in during the creation or allocation of another object **may** be accessed by the implementation any time that the created/allocated object is accessed unless explicitly stated otherwise. Device addresses and Vulkan objects passed to a recording command (**vkCmd***) **may** be accessed at any time during the execution of the command, when the command buffer is subsequently recorded into another command buffer, during any subsequent command that is recorded to either the command buffer or one it is recorded into, or while the command buffer is in the **pending state**, unless explicitly stated otherwise. Some additional operations hold references to other objects or application-owned memory beyond the duration of the command; in which case the access semantics and lifetime of those references are described by that command.

When destroying or freeing an object, implementations **must** not access any memory or other objects that **may** otherwise be accessed when the object is accessed. Applications **can** free or destroy objects in any order, except that parent objects **must** be freed only after all child objects are freed. An object is the parent of another child object if the parent was used as the first object parameter in the creation of the child. Once an object is freed or destroyed it **must** not be accessed again, either directly or via access through another object. Applications **must** not free or destroy any object while it is being accessed.

3.3.2. External Object Handles

As defined above, the scope of object handles created or allocated from a `VkDevice` is limited to that logical device. Objects which are not in scope are said to be external. To bring an external object into scope, an external handle **must** be exported from the object in the source scope and imported into the destination scope.

NOTE

The scope of external handles and their associated resources **may** vary according to their type, but they **can** generally be shared across process and API boundaries.

3.4. Application Binary Interface

The mechanism by which Vulkan is made available to applications is platform- or implementation-defined. On many platforms the C interface described in this Specification is provided by a shared library. Since shared libraries can be changed independently of the applications that use them, they present particular compatibility challenges, and this Specification places some requirements on them.

Shared library implementations **must** use the default Application Binary Interface (ABI) of the standard C compiler for the platform, or provide customized API headers that cause application code to use the implementation's non-default ABI. An ABI in this context means the size, alignment, and layout of C data types; the procedure calling convention; and the naming convention for shared library symbols corresponding to C functions. Customizing the calling convention for a platform is usually accomplished by defining [calling convention macros](#) appropriately in `vk_platform.h`.

On platforms where Vulkan is provided as a shared library, library symbols beginning with “vk” and followed by a digit or uppercase letter are reserved for use by the implementation. Applications which use Vulkan **must** not provide definitions of these symbols. This allows the Vulkan shared library to be updated with additional symbols for new API versions or extensions without causing symbol conflicts with existing applications.

Shared library implementations **should** provide library symbols for commands in the highest version of this Specification they support, and for [Window System Integration](#) extensions relevant to the platform. They **may** also provide library symbols for commands defined by additional extensions.

NOTE

These requirements and recommendations are intended to allow implementors to take advantage of platform-specific conventions for SDKs, ABIs, library versioning mechanisms, etc. while still minimizing the code changes necessary to port applications or libraries between platforms. Platform vendors, or providers of the *de facto* standard Vulkan shared library for a platform, are encouraged to document what symbols the shared library provides and how it will be versioned when new symbols are added.

Applications **should** only rely on shared library symbols for commands in the minimum core version required by the application. [vkGetInstanceProcAddr](#) and [vkGetDeviceProcAddr](#) **should** be used to obtain function pointers for commands in core versions beyond the application's minimum required version.

3.5. Command Syntax and Duration

The Specification describes Vulkan commands as functions or procedures using C99 syntax. Language bindings for other languages such as C++ and JavaScript **may** allow for stricter parameter passing, or object-oriented interfaces.

Vulkan uses the standard C types for the base type of scalar parameters (e.g. types from `<stdint.h>`), with exceptions described below, or elsewhere in the text when appropriate:

`VkBool32` represents boolean `True` and `False` values, since C does not have a sufficiently portable built-in boolean type:

```
// Provided by VK_VERSION_1_0
typedef uint32_t VkBool32;
```

`VK_TRUE` represents a boolean **True** (unsigned integer 1) value, and `VK_FALSE` a boolean **False** (unsigned integer 0) value.

All values returned from a Vulkan implementation in a `VkBool32` will be either `VK_TRUE` or `VK_FALSE`.

Applications **must** not pass any other values than `VK_TRUE` or `VK_FALSE` into a Vulkan implementation where a `VkBool32` is expected.

`VK_TRUE` is a constant representing a `VkBool32` **True** value.

```
#define VK_TRUE 1U
```

`VK_FALSE` is a constant representing a `VkBool32` **False** value.

```
#define VK_FALSE 0U
```

`VkDeviceSize` represents device memory size and offset values:

```
// Provided by VK_VERSION_1_0
typedef uint64_t VkDeviceSize;
```

`VkDeviceAddress` represents device buffer address values:

```
// Provided by VK_VERSION_1_0
typedef uint64_t VkDeviceAddress;
```

Valid Usage

- VUID-VkDeviceAddress-size-11364

A valid `VkDeviceAddress` **must** be equal to the sum of an address retrieved from a `VkBuffer` via `vkGetBufferDeviceAddress`, and any offset in the range `[0, size)`, where `size` is the value of `VkBufferCreateInfo::size` used to create that `VkBuffer`

- VUID-VkDeviceAddress-None-10894

If a `VkDeviceAddress` was retrieved from a non-sparse buffer, that buffer **must** be bound completely and contiguously to a single `VkDeviceMemory` object

Commands that create Vulkan objects are of the form `vkCreate*` and take `Vk*CreateInfo` structures with the parameters needed to create the object. These Vulkan objects are destroyed with commands of the form `vkDestroy*`.

The last in-parameter to each command that creates or destroys a Vulkan object is `pAllocator`. The `pAllocator` parameter **must** be `NULL`. Refer to the [Memory Allocation](#) chapter for further details.

Commands that allocate Vulkan objects owned by pool objects are of the form `vkAllocate*`, and take `Vk*AllocateInfo` structures. These Vulkan objects are freed with commands of the form `vkFree*`. These objects do not take allocators; if host memory is needed, they will use the allocator that was specified when their parent pool was created.

Commands are recorded into a command buffer by calling API commands of the form `vkCmd*`. Each such command **may** have different restrictions on where it **can** be used: in a primary and/or secondary command buffer, inside and/or outside a render pass, and in one or more of the supported queue types. These restrictions are documented together with the definition of each such command.

The *duration* of a Vulkan command refers to the interval between calling the command and its return to the caller.

3.5.1. Lifetime of Retrieved Results

Information is retrieved from the implementation with commands of the form `vkGet*` and `vkEnumerate*`.

Unless otherwise specified for an individual command, the results are *invariant*; that is, they will remain unchanged when retrieved again by calling the same command with the same parameters, so long as those parameters themselves all remain valid.

3.5.2. Array Results

Some query commands of the form `vkGet*` and `vkEnumerate*` enable retrieving multiple results in the form of a return array. Such commands typically have two pointer arguments as follows:

- An element count pointer pointing to an integer variable, conventionally named as `p*Count` where `*` is the capitalized singular form of the name of the retrieved values.
- A pointer to an array where the result array is retrieved, conventionally named as `p*` where `*` is the capitalized plural form of the name of the retrieved values.

If such commands are called with the array pointer set to `NULL`, then the number of retrievable

elements is returned in the variable pointed to by the element count pointer. Otherwise, the element count pointer **must** point to a variable set by the application to the number of elements in the return array, and on return the variable is overwritten with the number of elements actually written to the return array. If the input element count is less than the number of retrievable array elements, the query will write only as many elements to the return array as specified by the element count variable set by the application, and the command will return `VK_INCOMPLETE` instead of `VK_SUCCESS`, to indicate that not all retrievable array elements were returned.

NOTE

In practice, this means that applications will typically call such query commands twice:

- First, with the array pointer set to `NULL`, to retrieve the number of retrievable elements.
- Second, with the array pointer pointing to an application allocated storage for at least as many elements as indicated by the variable pointed to by the element count pointer, to retrieve at most as many of the retrievable elements.

Query commands that return one or more structures, regardless of whether they return a single or an array of structures with or without a `pNext` chain, **may** also contain arrays within those structures. Such return arrays are typically defined in the form of two members as follows:

- An integer value specifying the element count, conventionally named as `*Count` where `*` is the singular form of the name of the retrieved values.
- A pointer to an array where the result array is retrieved, conventionally named as `p*` where `*` is the capitalized plural form of the name of the retrieved values.

Analogously to query commands that return multiple results, if the command is called with the array pointer member of the output structure in question set to `NULL`, then the number of retrievable elements is returned in the element count member of that output structure. Otherwise, the element count **must** specify the number of elements in the return array, and on return the element count member is overwritten with the number of elements actually written to the return array. If the input element count is less than the number of retrievable array elements, the query will write only as many elements to the return array as specified by the input element count, and the command will return `VK_INCOMPLETE` instead of `VK_SUCCESS`, if the query command has a `VkResult` return type, to indicate that not all retrievable array elements were returned.

NOTE

Applications need to separately track the value they provided as the input element count member for such arrays and compare those with the returned element counts in order to determine whether the actually returned element count is smaller than the size of the return array. Another side effect of this is that it is impossible for the application to determine if the number of retrievable elements has increased beyond the provided input element count so using return arrays in output structures **should** be limited to *invariant* array results. In practice, this means that applications will typically call such query commands multiple times:

- First, with the array pointer member(s) set to `NULL`, to retrieve the number(s) of retrievable elements.

- Second, with the array pointer(s) pointing to an application allocated storage for at least as many elements as indicated by the element count member(s), to retrieve at most as many of the retrievable elements.
- Then the process **may** need to be repeated for all other newly introduced return arrays in any nested output structures indirectly specified through the previously retrieved result arrays.

Regardless of the type of query command, any array pointer member of an output structure **must** either be **NULL**, or point to an application-allocated array. Query commands **must** not return a pointer to implementation allocated storage in any output structure.

3.5.3. Opaque Binary Data Results

Some query commands of the form **vkGet*** retrieve opaque binary data in the form of a byte array and have a possible result code of **VK_ERROR_NOT_ENOUGH_SPACE_KHR**. Such commands always have two pointer arguments as follows:

- A binary data size pointer pointing to a **size_t** variable, conventionally named as **p*Size** where ***** is the capitalized form of the name of the retrieved binary data.
- A pointer to a byte array where the binary data is retrieved, conventionally named as **p*** where ***** is the capitalized form of the name of the retrieved binary data.

If such commands are called with the binary pointer not set to **NULL**, the binary size pointer **must** point to a variable set by the application to the allocated size of the binary pointer. These arguments **may** also be placed in an extensible structure, in which case the binary data size argument is not a pointer.

If the input binary size is less than the total retrievable binary size, the query will not write any data to the location pointed to the binary pointer, and the command will return **VK_ERROR_NOT_ENOUGH_SPACE_KHR** instead of **VK_SUCCESS**.

If the return code is **VK_SUCCESS** or **VK_ERROR_NOT_ENOUGH_SPACE_KHR**, the total size of the binary data that **can** be retrieved is returned in the variable pointed to by the binary size pointer.

If multiple binaries are being retrieved, **VK_ERROR_NOT_ENOUGH_SPACE_KHR** will be returned if any input binary sizes are less than their respective total retrievable binary sizes. Unless otherwise specified, this command will determine writing data to each binary individually based on if their input binary sizes are sufficiently sized, following the behavior for single binary retrieval.

For all other error codes, the contents of the return structures are undefined.

NOTE

If **VK_ERROR_NOT_ENOUGH_SPACE_KHR** is returned with a command that returns multiple binaries, the application **can** determine which binaries are undersized by comparing the total binary size that is returned for each binary against the allocated size that was provided to the command.

NOTE

Some binary queries do not behave consistently with this pattern for historical reasons, primarily that the **VK_ERROR_NOT_ENOUGH_SPACE_KHR** error code was not

defined until after those queries were written.

A NOTE is added to each such query, describing such inconsistent behavior.

3.6. Threading Behavior

Vulkan is intended to provide scalable performance when used on multiple host threads. All commands support being called concurrently from multiple threads, but certain parameters, or components of parameters are defined to be *externally synchronized*. This means that the caller **must** guarantee that no more than one thread is using such a parameter at a given time.

More precisely, Vulkan commands use simple stores to update the state of Vulkan objects. The implementation **may** not synchronize accesses to memory parameters or object parameters declared as externally synchronized with other accesses. If two commands access the same object or memory and at least one of the commands declares the object to be externally synchronized, then the caller **must** guarantee not only that the commands do not execute simultaneously, but also that the two commands are separated by an appropriate memory barrier (if needed). Similarly, if a Vulkan command accesses a non-const memory parameter and the application also accesses that memory, or if the application writes to that memory and the command accesses it as a const memory parameter, the application **must** ensure the accesses are properly synchronized with a memory barrier if needed.

NOTE

Memory barriers are particularly relevant for hosts based on the ARM CPU architecture, which is more weakly ordered than many developers are accustomed to from x86/x64 programming. Fortunately, most higher-level synchronization primitives (like the pthread library) perform memory barriers as a part of mutual exclusion, so mutexing Vulkan objects via these primitives will have the desired effect.

Any object parameters that are not labeled as externally synchronized are either not mutated by the command or are internally synchronized. Additionally, certain objects related to a command's parameters (e.g. command pools and descriptor pools) **may** be affected by a command, and **must** also be externally synchronized. These implicit parameters are documented as described below.

Parameters of commands that are externally synchronized are listed below.

Externally Synchronized Parameters and Members

- The **instance** parameter in [vkDestroyInstance](#)
- The **device** parameter in [vkDestroyDevice](#)
- The **queue** parameter in [vkQueueSubmit](#)
- The **fence** parameter in [vkQueueSubmit](#)
- The **queue** parameter in [vkQueueWaitIdle](#)
- The **memory** parameter in [vkMapMemory](#)
- The **memory** parameter in [vkUnmapMemory](#)

- The `buffer` parameter in `vkBindBufferMemory`
- The `image` parameter in `vkBindImageMemory`
- The `fence` parameter in `vkDestroyFence`
- The `semaphore` parameter in `vkDestroySemaphore`
- The `buffer` parameter in `vkDestroyBuffer`
- The `image` parameter in `vkDestroyImage`
- The `imageView` parameter in `vkDestroyImageView`
- The `commandPool` parameter in `vkResetCommandPool`
- The `commandPool` parameter in `vkFreeCommandBuffers`
- The `commandBuffer` parameter in `vkBeginCommandBuffer`
- The `commandBuffer` parameter in `vkEndCommandBuffer`
- The `commandBuffer` parameter in `vkResetCommandBuffer`
- The `commandBuffer` parameter in `vkCmdCopyBuffer`
- The `commandBuffer` parameter in `vkCmdCopyImage`
- The `commandBuffer` parameter in `vkCmdCopyBufferToImage`
- The `commandBuffer` parameter in `vkCmdCopyImageToBuffer`
- The `commandBuffer` parameter in `vkCmdUpdateBuffer`
- The `commandBuffer` parameter in `vkCmdFillBuffer`
- The `commandBuffer` parameter in `vkCmdPipelineBarrier`
- The `commandBuffer` parameter in `vkCmdBeginQuery`
- The `commandBuffer` parameter in `vkCmdEndQuery`
- The `commandBuffer` parameter in `vkCmdResetQueryPool`
- The `commandBuffer` parameter in `vkCmdWriteTimestamp`
- The `commandBuffer` parameter in `vkCmdCopyQueryPoolResults`
- The `commandBuffer` parameter in `vkCmdExecuteCommands`
- The `event` parameter in `vkDestroyEvent`
- The `event` parameter in `vkSetEvent`
- The `event` parameter in `vkResetEvent`
- The `bufferView` parameter in `vkDestroyBufferView`
- The `pipelineCache` parameter in `vkDestroyPipelineCache`
- The `pipelineCache` parameter in `vkCreateComputePipelines`, conditionally¹
- The `pipeline` parameter in `vkDestroyPipeline`
- The `pipelineLayout` parameter in `vkDestroyPipelineLayout`
- The `sampler` parameter in `vkDestroySampler`
- The `descriptorSetLayout` parameter in `vkDestroyDescriptorSetLayout`

- The `descriptorPool` parameter in `vkResetDescriptorPool`
- The `descriptorPool` parameter in `vkFreeDescriptorSets`
- The `commandBuffer` parameter in `vkCmdBindPipeline`
- The `commandBuffer` parameter in `vkCmdBindDescriptorSets`
- The `commandBuffer` parameter in `vkCmdClearColorImage`
- The `commandBuffer` parameter in `vkCmdDispatch`
- The `commandBuffer` parameter in `vkCmdDispatchIndirect`
- The `commandBuffer` parameter in `vkCmdSetEvent`
- The `commandBuffer` parameter in `vkCmdResetEvent`
- The `commandBuffer` parameter in `vkCmdWaitEvents`
- The `commandBuffer` parameter in `vkCmdPushConstants`
- The `pipelineCache` parameter in `vkCreateGraphicsPipelines`, conditionally¹
- The `framebuffer` parameter in `vkDestroyFramebuffer`
- The `renderPass` parameter in `vkDestroyRenderPass`
- The `commandBuffer` parameter in `vkCmdSetViewport`
- The `commandBuffer` parameter in `vkCmdSetScissor`
- The `commandBuffer` parameter in `vkCmdSetLineWidth`
- The `commandBuffer` parameter in `vkCmdSetDepthBias`
- The `commandBuffer` parameter in `vkCmdSetBlendConstants`
- The `commandBuffer` parameter in `vkCmdSetDepthBounds`
- The `commandBuffer` parameter in `vkCmdSetStencilCompareMask`
- The `commandBuffer` parameter in `vkCmdSetStencilWriteMask`
- The `commandBuffer` parameter in `vkCmdSetStencilReference`
- The `commandBuffer` parameter in `vkCmdBindIndexBuffer`
- The `commandBuffer` parameter in `vkCmdBindVertexBuffers`
- The `commandBuffer` parameter in `vkCmdDraw`
- The `commandBuffer` parameter in `vkCmdDrawIndexed`
- The `commandBuffer` parameter in `vkCmdDrawIndirect`
- The `commandBuffer` parameter in `vkCmdDrawIndexedIndirect`
- The `commandBuffer` parameter in `vkCmdBlitImage`
- The `commandBuffer` parameter in `vkCmdClearDepthStencilImage`
- The `commandBuffer` parameter in `vkCmdClearAttachments`
- The `commandBuffer` parameter in `vkCmdResolveImage`
- The `commandBuffer` parameter in `vkCmdBeginRenderPass`
- The `commandBuffer` parameter in `vkCmdNextSubpass`

- The `commandBuffer` parameter in `vkCmdEndRenderPass`
- The `commandBuffer` parameter in `vkCmdSetDeviceMask`
- The `commandBuffer` parameter in `vkCmdDispatchBase`
- The `ycbcrConversion` parameter in `vkDestroySamplerYcbcrConversion`
- The `commandBuffer` parameter in `vkCmdDrawIndirectCount`
- The `commandBuffer` parameter in `vkCmdDrawIndexedIndirectCount`
- The `commandBuffer` parameter in `vkCmdBeginRenderPass2`
- The `commandBuffer` parameter in `vkCmdNextSubpass2`
- The `commandBuffer` parameter in `vkCmdEndRenderPass2`
- The `commandPool` parameter in `vkGetCommandPoolMemoryConsumption`
- The `commandBuffer` parameter in `vkGetCommandPoolMemoryConsumption`
- The `surface` parameter in `vkDestroySurfaceKHR`
- The `swapchain` parameter in `vkAcquireNextImageKHR`
- The `semaphore` parameter in `vkAcquireNextImageKHR`
- The `fence` parameter in `vkAcquireNextImageKHR`
- The `queue` parameter in `vkQueuePresentKHR`
- The `surface` parameter in `vkGetDeviceGroupSurfacePresentModesKHR`
- The `surface` parameter in `vkGetPhysicalDevicePresentRectanglesKHR`
- The `display` parameter in `vkCreateDisplayModeKHR`
- The `mode` parameter in `vkGetDisplayPlaneCapabilitiesKHR`
- The `swapchain` parameter in `vkGetSwapchainStatusKHR`
- The `commandBuffer` parameter in `vkCmdSetFragmentShadingRateKHR`
- The `commandBuffer` parameter in `vkCmdRefreshObjectsKHR`
- The `commandBuffer` parameter in `vkCmdSetEvent2KHR`
- The `commandBuffer` parameter in `vkCmdResetEvent2KHR`
- The `commandBuffer` parameter in `vkCmdWaitEvents2KHR`
- The `commandBuffer` parameter in `vkCmdPipelineBarrier2KHR`
- The `commandBuffer` parameter in `vkCmdWriteTimestamp2KHR`
- The `queue` parameter in `vkQueueSubmit2KHR`
- The `fence` parameter in `vkQueueSubmit2KHR`
- The `commandBuffer` parameter in `vkCmdCopyBuffer2KHR`
- The `commandBuffer` parameter in `vkCmdCopyImage2KHR`
- The `commandBuffer` parameter in `vkCmdCopyBufferToImage2KHR`
- The `commandBuffer` parameter in `vkCmdCopyImageToBuffer2KHR`
- The `commandBuffer` parameter in `vkCmdBlitImage2KHR`

- The `commandBuffer` parameter in `vkCmdResolveImage2KHR`
- The `commandBuffer` parameter in `vkCmdSetLineStippleKHR`
- The `commandBuffer` parameter in `vkCmdSetDiscardRectangleEXT`
- The `commandBuffer` parameter in `vkCmdSetDiscardRectangleEnableEXT`
- The `commandBuffer` parameter in `vkCmdSetDiscardRectangleModeEXT`
- The `objectHandle` member of the `pNameInfo` parameter in `vkSetDebugUtilsObjectNameEXT`
- The `queue` parameter in `vkQueueBeginDebugUtilsLabelEXT`
- The `queue` parameter in `vkQueueEndDebugUtilsLabelEXT`
- The `queue` parameter in `vkQueueInsertDebugUtilsLabelEXT`
- The `commandBuffer` parameter in `vkCmdBeginDebugUtilsLabelEXT`
- The `commandBuffer` parameter in `vkCmdEndDebugUtilsLabelEXT`
- The `commandBuffer` parameter in `vkCmdInsertDebugUtilsLabelEXT`
- The `messenger` parameter in `vkDestroyDebugUtilsMessengerEXT`
- The `commandBuffer` parameter in `vkCmdSetSampleLocationsEXT`
- The `commandBuffer` parameter in `vkCmdSetLineStippleEXT`
- The `commandBuffer` parameter in `vkCmdSetCullModeEXT`
- The `commandBuffer` parameter in `vkCmdSetFrontFaceEXT`
- The `commandBuffer` parameter in `vkCmdSetPrimitiveTopologyEXT`
- The `commandBuffer` parameter in `vkCmdSetViewportWithCountEXT`
- The `commandBuffer` parameter in `vkCmdSetScissorWithCountEXT`
- The `commandBuffer` parameter in `vkCmdBindVertexBuffers2EXT`
- The `commandBuffer` parameter in `vkCmdSetDepthTestEnableEXT`
- The `commandBuffer` parameter in `vkCmdSetDepthWriteEnableEXT`
- The `commandBuffer` parameter in `vkCmdSetDepthCompareOpEXT`
- The `commandBuffer` parameter in `vkCmdSetDepthBoundsTestEnableEXT`
- The `commandBuffer` parameter in `vkCmdSetStencilTestEnableEXT`
- The `commandBuffer` parameter in `vkCmdSetStencilOpEXT`
- The `commandBuffer` parameter in `vkCmdSetVertexInputEXT`
- The `commandBuffer` parameter in `vkCmdSetPatchControlPointsEXT`
- The `commandBuffer` parameter in `vkCmdSetRasterizerDiscardEnableEXT`
- The `commandBuffer` parameter in `vkCmdSetDepthBiasEnableEXT`
- The `commandBuffer` parameter in `vkCmdSetLogicOpEXT`
- The `commandBuffer` parameter in `vkCmdSetPrimitiveRestartEnableEXT`
- The `commandBuffer` parameter in `vkCmdSetColorWriteEnableEXT`
- The `commandPool` member of `VkCommandBufferAllocateInfo`

- The `dstSet` member of `VkCopyDescriptorSet`, conditionally¹
- The `descriptorPool` member of `VkDescriptorSetAllocateInfo`
- The `buffer` member of `VkBindBufferMemoryInfo`
- The `image` member of `VkBindImageMemoryInfo`
- The `surface` member of `VkSwapchainCreateInfoKHR`
- The `swapchain` member of `VkBindImageMemorySwapchainInfoKHR`
- The `swapchain` member of `VkAcquireNextImageInfoKHR`
- The `semaphore` member of `VkAcquireNextImageInfoKHR`
- The `fence` member of `VkAcquireNextImageInfoKHR`
- The `semaphore` member of `VkImportSemaphoreFdInfoKHR`
- The `fence` member of `VkImportFenceFdInfoKHR`
- The `mode` member of `VkDisplayPlaneInfo2KHR`
- The `objectHandle` member of `VkRefreshObjectKHR`
- The `objectHandle` member of `VkDebugUtilsObjectTagInfoEXT`
- The `fence` member of `VkImportFenceSciSyncInfoNV`
- The `semaphore` member of `VkImportSemaphoreSciSyncInfoNV`

¹ See Valid Usage language for this token for details.

There are also a few instances where a command **can** take in an application-allocated list whose contents are externally synchronized parameters. In these cases, the caller **must** guarantee that at most one thread is using a given element within the list at a given time. These parameters are listed below.

Externally Synchronized Parameter and Member Lists

- Each element of the `pFences` parameter in `vkResetFences`
- Each element of the `pCommandBuffers` parameter in `vkFreeCommandBuffers`
- Each element of the `pDescriptorSets` parameter in `vkFreeDescriptorSets`
- The `dstSet` member of each element of the `pDescriptorWrites` parameter in `vkUpdateDescriptorSets`, conditionally¹
- Each element of the `pWaitSemaphores` member of `VkPresentInfoKHR`
- Each element of the `pSwapchains` member of `VkPresentInfoKHR`

¹ See Valid Usage language for this token for details.

In addition, there are some implicit parameters that need to be externally synchronized. For example, when a `commandBuffer` parameter needs to be externally synchronized, it implies that the `commandPool` from which that command buffer was allocated also needs to be externally

synchronized. The implicit parameters and their associated object are listed below.

Implicit Externally Synchronized Parameters

- All `VkPhysicalDevice` objects enumerated from `instance` in `vkDestroyInstance`
- All `VkQueue` objects created from `device` in `vkDestroyDevice`
- All `VkQueue` objects created from `device` in `vkDeviceWaitIdle`
- The `VkCommandPool` that `commandBuffer` was allocated from in `vkBeginCommandBuffer`
- The `VkCommandPool` that `commandBuffer` was allocated from in `vkEndCommandBuffer`
- The `VkCommandPool` that `commandBuffer` was allocated from in `vkResetCommandBuffer`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdCopyBuffer`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdCopyImage`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdCopyBufferToImage`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdCopyImageToBuffer`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdUpdateBuffer`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdFillBuffer`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdPipelineBarrier`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdBeginQuery`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdEndQuery`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdResetQueryPool`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdWriteTimestamp`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdCopyQueryPoolResults`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdExecuteCommands`
- Any `VkDescriptorSet` objects allocated from `descriptorPool` in `vkResetDescriptorPool`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdBindPipeline`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdBindDescriptorSets`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdClearColorImage`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdDispatch`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdDispatchIndirect`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetEvent`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdResetEvent`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdWaitEvents`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdPushConstants`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetViewport`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetScissor`

- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetLineWidth`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetDepthBias`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetBlendConstants`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetDepthBounds`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetStencilCompareMask`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetStencilWriteMask`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetStencilReference`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdBindIndexBuffer`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdBindVertexBuffers`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdDraw`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdDrawIndexed`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdDrawIndirect`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdDrawIndexedIndirect`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdBlitImage`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdClearDepthStencilImage`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdClearAttachments`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdResolveImage`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdBeginRenderPass`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdNextSubpass`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdEndRenderPass`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetDeviceMask`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdDispatchBase`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdDrawIndirectCount`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdDrawIndexedIndirectCount`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdBeginRenderPass2`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdNextSubpass2`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdEndRenderPass2`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetFragmentShadingRateKHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdRefreshObjectsKHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetEvent2KHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdResetEvent2KHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdWaitEvents2KHR`

- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdPipelineBarrier2KHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdWriteTimestamp2KHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdCopyBuffer2KHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdCopyImage2KHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdCopyBufferToImage2KHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdCopyImageToBuffer2KHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdBlitImage2KHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdResolveImage2KHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetLineStippleKHR`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetDiscardRectangleEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetDiscardRectangleEnableEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetDiscardRectangleModeEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdBeginDebugUtilsLabelEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdEndDebugUtilsLabelEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdInsertDebugUtilsLabelEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetSampleLocationsEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetLineStippleEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetCullModeEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetFrontFaceEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetPrimitiveTopologyEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetViewportWithCountEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetScissorWithCountEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdBindVertexBuffers2EXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetDepthTestEnableEXT`

- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetDepthWriteEnableEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetDepthCompareOpEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetDepthBoundsTestEnableEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetStencilTestEnableEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetStencilOpEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetVertexInputEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetPatchControlPointsEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetRasterizerDiscardEnableEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetDepthBiasEnableEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetLogicOpEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetPrimitiveRestartEnableEXT`
- The `VkCommandPool` that `commandBuffer` was allocated from, in `vkCmdSetColorWriteEnableEXT`

3.7. Valid Usage

Valid usage defines a set of conditions which **must** be met in order to achieve well-defined runtime behavior in an application. These conditions depend only on Vulkan state, and the parameters or objects whose usage is constrained by the condition.

The core layer assumes applications are using the API correctly. Except as documented elsewhere in the Specification, the behavior of the core layer to an application using the API incorrectly is undefined, and **may** include program termination. However, implementations **must** ensure that incorrect usage by an application does not affect the integrity of the operating system, the Vulkan implementation, or other applications in the system using Vulkan. In particular, any guarantees made by an operating system about whether memory from one process **can** be visible to another process or not **must** not be violated by a Vulkan implementation for **any memory allocation**. Vulkan implementations are not **required** to make additional security or integrity guarantees beyond those provided by the OS unless explicitly directed by the application's use of a particular feature or extension.

NOTE

For instance, if an operating system guarantees that data in all its memory allocations are set to zero when newly allocated, the Vulkan implementation **must** make the same guarantees for any allocations it controls (e.g. `VkDeviceMemory`).

Similarly, if an operating system guarantees that use-after-free of host allocations will not result in values written by another process becoming visible, the same guarantees **must** be made by the Vulkan implementation for device memory.

If the `protectedMemory` feature is supported, the implementation provides additional guarantees when invalid usage occurs to prevent values in protected memory from being accessed or inferred outside of protected operations, as described in [Protected Memory Access Rules](#).

Some valid usage conditions have dependencies on runtime limits or feature availability. It is possible to validate these conditions against Vulkan's minimum supported values for these limits and features, or some subset of other known values.

Valid usage conditions do not cover conditions where well-defined behavior (including returning an error code) exists.

Valid usage conditions **should** apply to the command or structure where complete information about the condition would be known during execution of an application. This is such that a validation layer or linter **can** be written directly against these statements at the point they are specified.

NOTE

This does lead to some non-obvious places for valid usage statements. For instance, the valid values for a structure might depend on a separate value in the calling command. In this case, the structure itself will not reference this valid usage as it is impossible to determine validity from the structure that it is invalid - instead this valid usage would be attached to the calling command.

Another example is draw state - the state setters are independent, and can cause a legitimately invalid state configuration between draw calls; so the valid usage statements are attached to the place where all state needs to be valid - at the drawing command.

Valid usage conditions are described in a block labeled "Valid Usage" following each command or structure they apply to.

3.7.1. Usage Validation

Vulkan is a layered API. The lowest layer is the core Vulkan layer, as defined by this Specification. The application **can** use additional layers above the core for debugging, validation, and other purposes.

One of the core principles of Vulkan is that building and submitting command buffers **should** be highly efficient. Thus error checking and validation of state in the core layer is minimal, although more rigorous validation **can** be enabled through the use of layers.

Validation of correct API usage is left to validation layers. Applications **should** be developed with validation layers enabled, to help catch and eliminate errors.

3.7.2. Implicit Valid Usage

Some valid usage conditions apply to all commands and structures in the API, unless explicitly denoted otherwise for a specific command or structure. These conditions are considered *implicit*, and are described in a block labeled “Valid Usage (Implicit)” following each command or structure they apply to. Implicit valid usage conditions are described in detail below.

Valid Usage for Object Handles

Any input parameter to a command that is an object handle **must** be a valid object handle, unless otherwise specified. An object handle is valid if:

- It has been created or allocated by a previous, successful call to the API. Such calls are noted in the Specification.
- It has not been deleted or freed by a previous call to the API. Such calls are noted in the Specification.
- Any objects used by that object, either as part of creation or execution, **must** also be valid.

The reserved values `VK_NULL_HANDLE` and `NULL` **can** be used in place of valid non-dispatchable handles and dispatchable handles, respectively, when *explicitly called out in the Specification*. Any command that creates an object successfully **must** not return these values. It is valid to pass these values to `vkDestroy*` or `vkFree*` commands, which will silently ignore these values.

Valid Usage for Pointers

Any parameter that is a pointer **must** be a *valid pointer* only if it is explicitly called out by a Valid Usage statement.

A pointer is “valid” if it points at memory containing values of the number and type(s) expected by the command, and all fundamental types accessed through the pointer (e.g. as elements of an array or as members of a structure) satisfy the alignment requirements of the host processor.

Valid Usage for Strings

Any parameter that is a pointer to `char` **must** be a finite sequence of values terminated by a null character, or if *explicitly called out in the Specification*, **can** be `NULL`.

Strings specified as UTF-8 encoded **must** not contain invalid UTF-8 sequences. See [String Representation](#) for additional information about strings.

Valid Usage for Enumerated Types

Any parameter of an enumerated type **must** be a valid enumerator for that type. Use of an enumerator is valid if the following conditions are true:

- The enumerator is defined as part of the enumerated type.
- The enumerator is not a value suffixed with `_MAX_ENUM`.
 - This value exists only to ensure that C `enum` types are 32 bits in size and **must** not be used by applications.

- If the enumerant is used in a function that has a [VkInstance](#) as its first parameter and either:
 - it was added by a core version that is supported (as reported by [vkEnumerateInstanceVersion](#)) and the value of [VkApplicationInfo::apiVersion](#) is greater than or equal to the version that added it; or
 - it was added by an [instance extension](#) that was enabled for the instance.
- If the enumerant is used in a function that has a [VkPhysicalDevice](#) object as its first parameter and either:
 - it was added by a core version that is supported by that device (as reported by [VkPhysicalDeviceProperties::apiVersion](#));
 - it was added by an [instance extension](#) that was enabled for the instance; or
 - it was added by a [device extension](#) that is supported by that device.
- If the enumerant is used in a function that has any other dispatchable object as its first parameter and either:
 - it was added by a core version that is supported for the device (as reported by [VkPhysicalDeviceProperties::apiVersion](#)); or
 - it was added by a [device extension](#) that was enabled for the device.

Any enumerated type returned from a query command or otherwise output from Vulkan to the application **must** not have a reserved value. Reserved values are values not defined by any extension for that enumerated type.

NOTE

In some special cases, an enumerant is only meaningful if a feature defined by an extension is also enabled, as well as the extension itself. The global “valid enumerant” rule described here does not address such cases.

NOTE

This language is intended to accommodate cases such as “hidden” extensions known only to driver internals, or layers enabling extensions without knowledge of the application, without allowing return of values not defined by any extension.

NOTE

Application developers are encouraged to be careful when using [switch](#) statements with Vulkan API enums. This is because new extensions can add new values to existing enums. Using a [default:](#) statement within a [switch](#) may avoid future compilation issues.

This is particularly true for enums such as [VkDriverId](#), which may have values added that do not belong to a corresponding new extension.

Valid Usage for Flags

A collection of flags is represented by a bitmask using the type [VkFlags](#):

```
// Provided by VK_VERSION_1_0
typedef uint32_t VkFlags;
```

Bitmasks are passed to many commands and structures to compactly represent options, but `VkFlags` is not used directly in the API. Instead, a `Vk*Flags` type which is an alias of `VkFlags`, and whose name matches the corresponding `Vk*FlagBits` that are valid for that type, is used.

Any `Vk*Flags` member or parameter used in the API as an input **must** be a valid combination of bit flags. A valid combination is either zero or the bitwise OR of valid bit flags.

An individual bit flag is valid for a `Vk*Flags` type if it would be a [valid enumerant](#) when used with the equivalent `Vk*FlagBits` type, where the bits type is obtained by taking the flag type and replacing the trailing `Flags` with `FlagBits`. For example, a flag value of type `VkColorComponentFlags` **must** contain only bit flags defined by `VkColorComponentFlagBits`.

Any `Vk*Flags` member or parameter returned from a query command or otherwise output from Vulkan to the application **may** contain bit flags undefined in its corresponding `Vk*FlagBits` type. An application **cannot** rely on the state of these unspecified bits.

Only the low-order 31 bits (bit positions zero through 30) are available for use as flag bits.

NOTE

This restriction is due to poorly defined behavior by C compilers given a C enumerant value of `0x80000000`. In some cases adding this enumerant value may increase the size of the underlying `Vk*FlagBits` type, breaking the ABI.

A collection of 64-bit flags is represented by a bitmask using the type `VkFlags64`:

```
// Provided by VK_KHR_synchronization2
typedef uint64_t VkFlags64;
```

When the 31 bits available in `VkFlags` are insufficient, the `VkFlags64` type can be passed to commands and structures to represent up to 64 options. `VkFlags64` is not used directly in the API. Instead, a `Vk*Flags2` type which is an alias of `VkFlags64`, and whose name matches the corresponding `Vk*FlagBits2` that are valid for that type, is used.

Any `Vk*Flags2` member or parameter used in the API as an input **must** be a valid combination of bit flags. A valid combination is either zero or the bitwise OR of valid bit flags.

An individual bit flag is valid for a `Vk*Flags2` type if it would be a [valid enumerant](#) when used with the equivalent `Vk*FlagBits2` type, where the bits type is obtained by taking the flag type and replacing the trailing `Flags2` with `FlagBits2`. For example, a flag value of type `VkAccessFlags2KHR` **must** contain only bit flags defined by `VkAccessFlagBits2KHR`.

Any `Vk*Flags2` member or parameter returned from a query command or otherwise output from Vulkan to the application **may** contain bit flags undefined in its corresponding `Vk*FlagBits2` type. An application **cannot** rely on the state of these unspecified bits.

NOTE

Both the `Vk*FlagBits2` type, and the individual bits defined for that type, are defined as `uint64_t` integers in the C API. This is in contrast to the 32-bit types, where the `Vk*FlagBits` type is defined as a C `enum` and the individual bits as enumerants belonging to that `enum`. As a result, there is less compile time type checking possible

for the 64-bit types. This is unavoidable since there is no sufficiently portable way to define a 64-bit `enum` type in C99.

Valid Usage for Structure Types

Any parameter that is a structure containing a `sType` member **must** have a value of `sType` which is a valid `VkStructureType` value matching the type of the structure.

Valid Usage for Structure Pointer Chains

Any parameter that is a structure containing a `void* pNext` member **must** have a value of `pNext` that is either `NULL`, or is a pointer to a valid *extending structure*, containing `sType` and `pNext` members as described in the [Vulkan Documentation and Extensions](#) document in the section “Extending Structures”. The set of structures connected by `pNext` pointers is referred to as a *pNext chain*.

Each structure included in the `pNext` chain **must** be defined at runtime by either:

- a core version which is supported
- an extension which is enabled
- a supported device extension in the case of physical-device-level functionality added by the device extension

Each type of extending structure **must** not appear more than once in a `pNext` chain, including any [aliases](#). This general rule may be explicitly overridden for specific structures.

Any component of the implementation (the loader, any enabled layers, and drivers) **must** skip over, without processing (other than reading the `sType` and `pNext` members) any extending structures in the chain not defined by core versions or extensions supported by that component.

As a convenience to implementations and layers needing to iterate through a structure pointer chain, the Vulkan API provides two *base structures*. These structures allow for some type safety, and can be used by Vulkan API functions that operate on generic inputs and outputs.

The `VkBaseInStructure` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBaseInStructure {
    VkStructureType      sType;
    const struct VkBaseInStructure* pNext;
} VkBaseInStructure;
```

- `sType` is the structure type of the structure being iterated through.
- `pNext` is `NULL` or a pointer to the next structure in a structure chain.

`VkBaseInStructure` can be used to facilitate iterating through a read-only structure pointer chain.

The `VkBaseOutStructure` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBaseOutStructure {
    VkStructureType      sType;
    struct VkBaseOutStructure* pNext;
} VkBaseOutStructure;
```

- **sType** is the structure type of the structure being iterated through.
- **pNext** is **NULL** or a pointer to the next structure in a structure chain.

VkBaseOutStructure can be used to facilitate iterating through a structure pointer chain that returns data back to the application.

Valid Usage for Nested Structures

The above conditions also apply recursively to members of structures provided as input to a command, either as a direct argument to the command, or themselves a member of another structure.

Specifics on valid usage of each command are covered in their individual sections.

Valid Usage for Extensions

Instance-level functionality or behavior added by an [instance extension](#) to the API **must** not be used unless that extension is supported by the instance as determined by [vkEnumerateInstanceExtensionProperties](#), and that extension is enabled in [VkInstanceCreateInfo](#).

Physical-device-level functionality or behavior added by an [instance extension](#) to the API **must** not be used unless that extension is supported by the instance as determined by [vkEnumerateInstanceExtensionProperties](#), and that extension is enabled in [VkInstanceCreateInfo](#).

Physical-device-level functionality or behavior added by a [device extension](#) to the API **must** not be used unless the conditions described in [Extending Physical Device From Device Extensions](#) are met.

Device-level functionality added by a [device extension](#) that is dispatched from a [VkDevice](#), or from a child object of a [VkDevice](#) **must** not be used unless that extension is supported by the device as determined by [vkEnumerateDeviceExtensionProperties](#), and that extension is enabled in [VkDeviceCreateInfo](#).

Valid Usage for Newer Core Versions

Instance-level functionality or behavior added by a [new core version](#) of the API **must** not be used unless it is supported by the instance as determined by [vkEnumerateInstanceVersion](#) and the specified version of [VkApplicationInfo::apiVersion](#).

Physical-device-level functionality or behavior added by a [new core version](#) of the API **must** not be used unless it is supported by the physical device as determined by [VkPhysicalDeviceProperties::apiVersion](#) and the specified version of [VkApplicationInfo::apiVersion](#).

Device-level functionality or behavior added by a [new core version](#) of the API **must** not be used

unless it is supported by the device as determined by `VkPhysicalDeviceProperties::apiVersion` and the specified version of `VkApplicationInfo::apiVersion`.

3.8. `VkResult` Return Codes

While the core Vulkan API is not designed to capture incorrect usage, some circumstances still require return codes. Commands in Vulkan return their status via return codes that are in one of two categories:

- Successful completion codes are returned when a command needs to communicate success or status information. All successful completion codes are non-negative values.
- Runtime error codes are returned when a command needs to communicate a failure that could only be detected at runtime. All runtime error codes are negative values.

All return codes in Vulkan are reported via `VkResult` return values. The possible codes are:

```
// Provided by VK_VERSION_1_0
typedef enum VkResult {
    VK_SUCCESS = 0,
    VK_NOT_READY = 1,
    VK_TIMEOUT = 2,
    VK_EVENT_SET = 3,
    VK_EVENT_RESET = 4,
    VK_INCOMPLETE = 5,
    VK_ERROR_OUT_OF_HOST_MEMORY = -1,
    VK_ERROR_OUT_OF_DEVICE_MEMORY = -2,
    VK_ERROR_INITIALIZATION_FAILED = -3,
    VK_ERROR_DEVICE_LOST = -4,
    VK_ERROR_MEMORY_MAP_FAILED = -5,
    VK_ERROR_LAYER_NOT_PRESENT = -6,
    VK_ERROR_EXTENSION_NOT_PRESENT = -7,
    VK_ERROR_FEATURE_NOT_PRESENT = -8,
    VK_ERROR_INCOMPATIBLE_DRIVER = -9,
    VK_ERROR_TOO_MANY_OBJECTS = -10,
    VK_ERROR_FORMAT_NOT_SUPPORTED = -11,
    VK_ERROR_FRAGMENTED_POOL = -12,
    VK_ERROR_UNKNOWN = -13,
// Provided by VK_VERSION_1_0
    VK_ERROR_VALIDATION_FAILED = -1000011001,
// Provided by VK_VERSION_1_1
    VK_ERROR_OUT_OF_POOL_MEMORY = -1000069000,
// Provided by VK_VERSION_1_1
    VK_ERROR_INVALID_EXTERNAL_HANDLE = -1000072003,
// Provided by VK_VERSION_1_2
    VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS = -1000257000,
// Provided by VK_VERSION_1_2
    VK_ERROR_FRAGMENTATION = -1000161000,
    VK_ERROR_NOT_PERMITTED = -1000174001,
// Provided by VKSC_VERSION_1_0
```

```

    VK_ERROR_INVALID_PIPELINE_CACHE_DATA = -1000298000,
// Provided by VKSC_VERSION_1_0
    VK_ERROR_NO_PIPELINE_MATCH = -1000298001,
// Provided by VK_KHR_surface
    VK_ERROR_SURFACE_LOST_KHR = -1000000000,
// Provided by VK_KHR_surface
    VK_ERROR_NATIVE_WINDOW_IN_USE_KHR = -1000000001,
// Provided by VK_KHR_swapchain
    VK_SUBOPTIMAL_KHR = 1000001003,
// Provided by VK_KHR_swapchain
    VK_ERROR_OUT_OF_DATE_KHR = -1000001004,
// Provided by VK_KHR_display_swapchain
    VK_ERROR_INCOMPATIBLE_DISPLAY_KHR = -1000003001,
// Provided by VK_EXT_image_drm_format_modifier
    VK_ERROR_INVALID_DRM_FORMAT_MODIFIER_PLANE_LAYOUT_EXT = -1000158000,
// Provided by VK_KHR_global_priority
    VK_ERROR_NOT_PERMITTED_KHR = VK_ERROR_NOT_PERMITTED,
} VkResult;

```

Success Codes

- **VK_SUCCESS** Command successfully completed
- **VK_NOT_READY** A fence or query has not yet completed
- **VK_TIMEOUT** A wait operation has not completed in the specified time
- **VK_EVENT_SET** An event is signaled
- **VK_EVENT_RESET** An event is unsignaled
- **VK_INCOMPLETE** A return array was too small for the result
- **VK_SUBOPTIMAL_KHR** A swapchain no longer matches the surface properties exactly, but **can** still be used to present to the surface successfully.

Error Codes

- **VK_ERROR_OUT_OF_HOST_MEMORY** A host memory allocation has failed.
- **VK_ERROR_OUT_OF_DEVICE_MEMORY** A device memory allocation has failed.
- **VK_ERROR_INITIALIZATION_FAILED** Initialization of an object could not be completed for implementation-specific reasons.
- **VK_ERROR_DEVICE_LOST** The logical or physical device has been lost. See [Lost Device](#)
- **VK_ERROR_MEMORY_MAP_FAILED** Mapping of a memory object has failed.
- **VK_ERROR_LAYER_NOT_PRESENT** A requested layer is not present or could not be loaded.
- **VK_ERROR_EXTENSION_NOT_PRESENT** A requested extension is not supported.
- **VK_ERROR_FEATURE_NOT_PRESENT** A requested feature is not supported.
- **VK_ERROR_INCOMPATIBLE_DRIVER** The requested version of Vulkan is not supported by the driver or is otherwise incompatible for implementation-specific reasons.
- **VK_ERROR_TOO_MANY_OBJECTS** Too many objects of the type have already been created.

- **VK_ERROR_FORMAT_NOT_SUPPORTED** A requested format is not supported on this device.
- **VK_ERROR_FRAGMENTED_POOL** A pool allocation has failed due to fragmentation of the pool's memory. This **must** only be returned if no attempt to allocate host or device memory was made to accommodate the new allocation. This **should** be returned in preference to **VK_ERROR_OUT_OF_POOL_MEMORY**, but only if the implementation is certain that the pool allocation failure was due to fragmentation.
- **VK_ERROR_SURFACE_LOST_KHR** A surface is no longer available.
- **VK_ERROR_NATIVE_WINDOW_IN_USE_KHR** The requested window is already in use by Vulkan or another API in a manner which prevents it from being used again.
- **VK_ERROR_OUT_OF_DATE_KHR** A surface has changed in such a way that it is no longer compatible with the swapchain, and further presentation requests using the swapchain will fail. Applications **must** query the new surface properties and recreate their swapchain if they wish to continue presenting to the surface.
- **VK_ERROR_INCOMPATIBLE_DISPLAY_KHR** The display used by a swapchain does not use the same presentable image layout, or is incompatible in a way that prevents sharing an image.
- **VK_ERROR_OUT_OF_POOL_MEMORY** A pool memory allocation has failed. This **must** only be returned if no attempt to allocate host or device memory was made to accommodate the new allocation. If the failure was definitely due to fragmentation of the pool, **VK_ERROR_FRAGMENTED_POOL** **should** be returned instead.
- **VK_ERROR_INVALID_EXTERNAL_HANDLE** An external handle is not a valid handle of the specified type.
- **VK_ERROR_FRAGMENTATION** A descriptor pool creation has failed due to fragmentation.
- **VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS** A buffer creation or memory allocation failed because the requested address is not available.
- **VK_ERROR_VALIDATION_FAILED** A command failed because invalid usage was detected by the implementation or a validation layer. This **may** result in the command not being dispatched to the ICD.
- **VK_ERROR_INVALID_PIPELINE_CACHE_DATA** The supplied pipeline cache data was not valid for the current implementation.
- **VK_ERROR_NO_PIPELINE_MATCH** The implementation did not find a match in the pipeline cache for the specified pipeline, or **VkPipelineOfflineCreateInfo** was not provided to the **vkCreate*Pipelines** function.
- **VK_ERROR_NOT_PERMITTED** The driver implementation has denied a request to acquire a priority above the default priority (**VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_EXT**) because the application does not have sufficient privileges.
- **VK_ERROR_UNKNOWN** An unknown error has occurred; either the application has provided invalid input, or an implementation failure has occurred.

If a command returns a runtime error, unless otherwise specified any output parameters will have undefined contents, except that if the output parameter is a structure with **sType** and **pNext** fields, those fields will be unmodified. Any structures chained from **pNext** will also have undefined contents, except that **sType** and **pNext** will be unmodified.

VK_ERROR_OUT_OF_*_MEMORY errors do not modify any currently existing Vulkan objects. Objects that

have already been successfully created **can** still be used by the application. If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `VK_ERROR_OUT_OF_HOST_MEMORY` **must** not be returned from any physical or logical device command which explicitly disallows it.

NOTE

As a general rule, `Free`, `Release`, and `Reset` commands do not return `VK_ERROR_OUT_OF_HOST_MEMORY`, while any other command with a return code **may** return it. Any exceptions from this rule are described for those commands.

`VK_ERROR_UNKNOWN` will be returned by an implementation when an unexpected error occurs that cannot be attributed to valid behavior of the application and implementation. Under these conditions, it **may** be returned from any command returning a `VkResult`.

NOTE

`VK_ERROR_UNKNOWN` is not expected to ever be returned if the application behavior is valid, and if the implementation is bug-free. If `VK_ERROR_UNKNOWN` is returned, the application should be checked against the latest validation layers to verify correct behavior as much as possible. If no issues are identified it could be an implementation issue, and the implementor should be contacted for support.

Any command returning a `VkResult` **may** return `VK_ERROR_VALIDATION_FAILED` if a violation of valid usage is detected.

Performance-critical commands generally do not have return codes. If a runtime error occurs in such commands, the implementation will defer reporting the error until a specified point. For commands that record into command buffers (`vkCmd*`) runtime errors are reported by `vkEndCommandBuffer`.

NOTE

Implementations can also use [Fault Handling](#) to report runtime errors where suitable return values are not available or to provide more prompt notification of an error.

3.9. Numeric Representation and Computation

Implementations normally perform computations in floating-point, and **must** meet the range and precision requirements defined under “Floating-Point Computation” below.

These requirements only apply to computations performed in Vulkan operations outside of shader execution, such as texture image specification and sampling, and per-fragment operations. Range and precision requirements during shader execution differ and are specified by the [Precision and Operation of SPIR-V Instructions](#) section.

In some cases, the representation and/or precision of operations is implicitly limited by the specified format of vertex or texel data consumed by Vulkan. Specific floating-point formats are described later in this section.

3.9.1. Floating-Point Computation

Most floating-point computation is performed in SPIR-V shader modules. The properties of

computation within shaders are constrained as defined by the [Precision and Operation of SPIR-V Instructions](#) section.

Some floating-point computation is performed outside of shaders, such as viewport and depth range calculations. For these computations, we do not specify how floating-point numbers are to be represented, or the details of how operations on them are performed, but only place minimal requirements on representation and precision as described in the remainder of this section.

We require simply that numbers' floating-point parts contain enough bits and that their exponent fields are large enough so that individual results of floating-point operations are accurate to about 1 part in 10^5 . The maximum representable magnitude for all floating-point values **must** be at least 2^{32} .

$x \times 0 = 0 \times x = 0$ for any non-infinite and non-NaN x .

$1 \times x = x \times 1 = x$.

$x + 0 = 0 + x = x$.

$0^0 = 1$.

Occasionally, further requirements will be specified. Most single-precision floating-point formats meet these requirements.

The special values Inf and -Inf encode values with magnitudes too large to be represented; the special value NaN encodes “Not A Number” values resulting from undefined arithmetic operations such as $0 / 0$. Implementations **may** support Inf and NaN in their floating-point computations. Any computation which does not support either Inf or NaN, for which that value is an input or output will yield an undefined value.

3.9.2. Floating-Point Format Conversions

When a value is converted to a defined floating-point representation, finite values falling between two representable finite values are rounded to one or the other. The rounding mode is not defined. Finite values whose magnitude is larger than that of any representable finite value may be rounded either to the closest representable finite value or to the appropriately signed infinity. For unsigned destination formats any negative values are converted to zero. Positive infinity is converted to positive infinity; negative infinity is converted to negative infinity in signed formats and to zero in unsigned formats; and any NaN is converted to a NaN.

3.9.3. 16-Bit Floating-Point Numbers

16-bit floating-point numbers are defined in the “16-bit floating-point numbers” section of the [Khronos Data Format Specification](#).

3.9.4. Unsigned 11-Bit Floating-Point Numbers

Unsigned 11-bit floating-point numbers are defined in the “Unsigned 11-bit floating-point numbers” section of the [Khronos Data Format Specification](#).

3.9.5. Unsigned 10-Bit Floating-Point Numbers

Unsigned 10-bit floating-point numbers are defined in the “Unsigned 10-bit floating-point numbers” section of the [Khronos Data Format Specification](#).

3.9.6. 8-bit booleans

An 8-bit boolean uses the following representation for `true` and `false`:

- Zero to represent `false`
- Any non-zero value to represent `true`

3.9.7. General Requirements

Any representable floating-point value in the appropriate format is legal as input to a Vulkan command that requires floating-point data. The result of providing a value that is not a floating-point number to such a command is unspecified, but **must** not lead to Vulkan interruption or termination. For example, providing a negative zero (where applicable) or a denormalized number to a Vulkan command **must** yield deterministic results, while providing a NaN or Inf yields unspecified results.

Some calculations require division. In such cases (including implied divisions performed by vector normalization), division by zero produces an unspecified result but **must** not lead to Vulkan interruption or termination.

3.10. Fixed-Point Data Conversions

When generic vertex attributes and pixel color or depth *components* are represented as integers, they are often (but not always) considered to be *normalized*. Normalized integer values are treated specially when being converted to and from floating-point values, and are usually referred to as *normalized fixed-point*.

In the remainder of this section, *b* denotes the bit width of the fixed-point integer representation. When the integer is one of the types defined by the API, *b* is the bit width of that type. When the integer comes from an [image](#) containing color or depth component texels, *b* is the number of bits allocated to that component in its [specified image format](#).

The signed and unsigned fixed-point representations are assumed to be *b*-bit binary two's-complement integers and binary unsigned integers, respectively.

3.10.1. Conversion From Normalized Fixed-Point to Floating-Point

Unsigned normalized fixed-point integers represent numbers in the range [0,1]. The conversion from an unsigned normalized fixed-point value *c* to the corresponding floating-point value *f* is

defined as

$$f = \frac{c}{2^b - 1}$$

Signed normalized fixed-point integers represent numbers in the range [-1,1]. The conversion from a signed normalized fixed-point value c to the corresponding floating-point value f is performed using

$$f = \max\left(\frac{c}{2^{b-1} - 1}, -1.0\right)$$

Only the range $[-2^{b-1} + 1, 2^{b-1} - 1]$ is used to represent signed fixed-point values in the range [-1,1]. For example, if $b = 8$, then the integer value -127 corresponds to -1.0 and the value 127 corresponds to 1.0. This equation is used everywhere that signed normalized fixed-point values are converted to floating-point.

Note that while zero is exactly expressible in this representation, one value (-128 in the example) is outside the representable range, and implementations **must** clamp it to -1.0. Where the value is subject to further processing by the implementation, e.g. during texture filtering, values less than -1.0 **may** be used but the result **must** be clamped before the value is returned to shaders.

3.10.2. Conversion From Floating-Point to Normalized Fixed-Point

The conversion from a floating-point value f to the corresponding unsigned normalized fixed-point value c is defined by first clamping f to the range [0,1], then computing

$$c = \text{convertFloatToUint}(f \times (2^b - 1), b)$$

where $\text{convertFloatToUint}(r, b)$ returns one of the two unsigned binary integer values with exactly b bits which are closest to the floating-point value r . Implementations **should** round to nearest. If r is equal to an integer, then that integer value **must** be returned. In particular, if f is equal to 0.0 or 1.0, then c **must** be assigned 0 or $2^b - 1$, respectively.

The conversion from a floating-point value f to the corresponding signed normalized fixed-point value c is performed by clamping f to the range [-1,1], then computing

$$c = \text{convertFloatToInt}(f \times (2^{b-1} - 1), b)$$

where $\text{convertFloatToInt}(r, b)$ returns one of the two signed two's-complement binary integer values with exactly b bits which are closest to the floating-point value r . Implementations **should** round to nearest. If r is equal to an integer, then that integer value **must** be returned. In particular, if f is equal to -1.0, 0.0, or 1.0, then c **must** be assigned $-(2^{b-1} - 1)$, 0, or $2^{b-1} - 1$, respectively.

This equation is used everywhere that floating-point values are converted to signed normalized fixed-point.

3.11. String Representation

Strings passed into and returned from Vulkan API commands are usually defined to be null-terminated and UTF-8 encoded.

NOTE

Exceptions to this rule exist only when strings are defined or used by operating system APIs where that OS has a different convention. For example, `VkExportMemoryWin32HandleInfoKHR::name` is a null-terminated UTF-16 encoded string used in conjunction with Windows handles.

When a UTF-8 string is **returned from** a Vulkan API query, it is returned in a fixed-length buffer of C `char`. For example, a string returned in `VkPhysicalDeviceProperties::deviceName` has maximum length `VK_MAX_PHYSICAL_DEVICE_NAME_SIZE`, and a string returned in `VkExtensionProperties::extensionName` has maximum length `VK_MAX_EXTENSION_NAME_SIZE`. The string, **including** its null terminator, will always fit completely within this buffer. If the string is shorter than the buffer size, the contents of `char` in the buffer following the null terminator are undefined.

When a UTF-8 string is **passed into** a Vulkan API, such as `VkDeviceCreateInfo::ppEnabledExtensionNames`, there is no explicit limit on the length of that string. However, the string **must** contain a valid UTF-8 encoded string and **must** be null-terminated.

3.12. Common Object Types

Some types of Vulkan objects are used in many different structures and command parameters, and are described here. These types include *offsets*, *extents*, and *rectangles*.

3.12.1. Offsets

Offsets are used to describe a pixel location within an image or framebuffer, as an (x,y) location for two-dimensional images, or an (x,y,z) location for three-dimensional images.

A two-dimensional offset is defined by the structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkOffset2D {
    int32_t    x;
    int32_t    y;
} VkOffset2D;
```

- `x` is the x offset.
- `y` is the y offset.

A three-dimensional offset is defined by the structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkOffset3D {
    int32_t    x;
    int32_t    y;
    int32_t    z;
} VkOffset3D;
```

- **x** is the x offset.
- **y** is the y offset.
- **z** is the z offset.

3.12.2. Extents

Extents are used to describe the size of a rectangular region of pixels within an image or framebuffer, as (width,height) for two-dimensional images, or as (width,height,depth) for three-dimensional images.

A two-dimensional extent is defined by the structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkExtent2D {
    uint32_t    width;
    uint32_t    height;
} VkExtent2D;
```

- **width** is the width of the extent.
- **height** is the height of the extent.

A three-dimensional extent is defined by the structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkExtent3D {
    uint32_t    width;
    uint32_t    height;
    uint32_t    depth;
} VkExtent3D;
```

- **width** is the width of the extent.
- **height** is the height of the extent.
- **depth** is the depth of the extent.

3.12.3. Rectangles

Rectangles are used to describe a specified rectangular region of pixels within an image or framebuffer. Rectangles include both an offset and an extent of the same dimensionality, as

described above. Two-dimensional rectangles are defined by the structure

```
// Provided by VK_VERSION_1_0
typedef struct VkRect2D {
    VkOffset2D    offset;
    VkExtent2D    extent;
} VkRect2D;
```

- `offset` is a `VkOffset2D` specifying the rectangle offset.
- `extent` is a `VkExtent2D` specifying the rectangle extent.

3.12.4. Structure Types

Each value corresponds to a particular structure with a `sType` member with a matching name. As a general rule, the name of each `VkStructureType` value is obtained by taking the name of the structure, stripping the leading `Vk`, prefixing each capital letter with `_`, converting the entire resulting string to upper case, and prefixing it with `VK_STRUCTURE_TYPE_`. For example, structures of type `VkImageCreateInfo` correspond to a `VkStructureType` value of `VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO`, and thus a structure of this type **must** have its `sType` member set to this value before it is passed to the API.

The values `VK_STRUCTURE_TYPE_LOADER_INSTANCE_CREATE_INFO` and `VK_STRUCTURE_TYPE_LOADER_DEVICE_CREATE_INFO` are reserved for internal use by the loader, and do not have corresponding Vulkan structures in this Specification.

Structure types supported by the Vulkan API include:

```
// Provided by VK_VERSION_1_0
typedef enum VkStructureType {
    VK_STRUCTURE_TYPE_APPLICATION_INFO = 0,
    VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO = 1,
    VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO = 2,
    VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO = 3,
    VK_STRUCTURE_TYPE_SUBMIT_INFO = 4,
    VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO = 5,
    VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE = 6,
    VK_STRUCTURE_TYPE_FENCE_CREATE_INFO = 8,
    VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO = 9,
    VK_STRUCTURE_TYPE_EVENT_CREATE_INFO = 10,
    VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO = 11,
    VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO = 12,
    VK_STRUCTURE_TYPE_BUFFER_VIEW_CREATE_INFO = 13,
    VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO = 14,
    VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO = 15,
    VK_STRUCTURE_TYPE_PIPELINE_CACHE_CREATE_INFO = 17,
    VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO = 18,
    VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO = 19,
    VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO = 20,
```

```

VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_STATE_CREATE_INFO = 21,
VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO = 22,
VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO = 23,
VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO = 24,
VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO = 25,
VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO = 26,
VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO = 27,
VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO = 28,
VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_CREATE_INFO = 29,
VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO = 30,
VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO = 31,
VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO = 32,
VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO = 33,
VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO = 34,
VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET = 35,
VK_STRUCTURE_TYPE_COPY_DESCRIPTOR_SET = 36,
VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO = 37,
VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO = 38,
VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO = 39,
VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO = 40,
VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_INFO = 41,
VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO = 42,
VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO = 43,
VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER = 44,
VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER = 45,
VK_STRUCTURE_TYPE_MEMORY_BARRIER = 46,
VK_STRUCTURE_TYPE_LOADER_INSTANCE_CREATE_INFO = 47,
VK_STRUCTURE_TYPE_LOADER_DEVICE_CREATE_INFO = 48,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_INFO = 1000157000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_INFO = 1000157001,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_MEMORY_DEDICATED_REQUIREMENTS = 1000127000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_MEMORY_DEDICATED_ALLOCATE_INFO = 1000127001,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_FLAGS_INFO = 1000060000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_DEVICE_GROUP_COMMAND_BUFFER_BEGIN_INFO = 1000060004,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_DEVICE_GROUP_SUBMIT_INFO = 1000060005,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_DEVICE_GROUP_INFO = 1000060013,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_DEVICE_GROUP_INFO = 1000060014,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GROUP_PROPERTIES = 1000070000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_DEVICE_GROUP_DEVICE_CREATE_INFO = 1000070001,
// Provided by VK_VERSION_1_1

```

```
VK_STRUCTURE_TYPE_BUFFER_MEMORY_REQUIREMENTS_INFO_2 = 1000146000,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_IMAGE_MEMORY_REQUIREMENTS_INFO_2 = 1000146001,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_MEMORY_REQUIREMENTS_2 = 1000146003,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FEATURES_2 = 1000059000,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROPERTIES_2 = 1000059001,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_2 = 1000059002,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_IMAGE_FORMAT_PROPERTIES_2 = 1000059003,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_FORMAT_INFO_2 = 1000059004,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_QUEUE_FAMILY_PROPERTIES_2 = 1000059005,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PROPERTIES_2 = 1000059006,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_IMAGE_VIEW_USAGE_CREATE_INFO = 1000117002,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_PROTECTED_SUBMIT_INFO = 1000145000,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROTECTED_MEMORY_FEATURES = 1000145001,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROTECTED_MEMORY_PROPERTIES = 1000145002,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_DEVICE_QUEUE_INFO_2 = 1000145003,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_IMAGE_FORMAT_INFO = 1000071000,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_EXTERNAL_IMAGE_FORMAT_PROPERTIES = 1000071001,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_BUFFER_INFO = 1000071002,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_EXTERNAL_BUFFER_PROPERTIES = 1000071003,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ID_PROPERTIES = 1000071004,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_BUFFER_CREATE_INFO = 1000072000,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO = 1000072001,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO = 1000072002,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FENCE_INFO = 1000112000,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_EXTERNAL_FENCE_PROPERTIES = 1000112001,  
// Provided by VK_VERSION_1_1  
VK_STRUCTURE_TYPE_EXPORT_FENCE_CREATE_INFO = 1000113000,
```

```

// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_CREATE_INFO = 1000077000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SEMAPHORE_INFO = 1000076000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_EXTERNAL_SEMAPHORE_PROPERTIES = 1000076001,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_PROPERTIES = 1000094000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_16BIT_STORAGE_FEATURES = 1000083000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTERS_FEATURES = 1000120000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_3_PROPERTIES = 1000168000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_SUPPORT = 1000168001,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_CREATE_INFO = 1000156000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_INFO = 1000156001,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_BIND_IMAGE_PLANE_MEMORY_INFO = 1000156002,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_IMAGE_PLANE_MEMORY_REQUIREMENTS_INFO = 1000156003,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_YCBCR_CONVERSION_FEATURES = 1000156004,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_IMAGE_FORMAT_PROPERTIES = 1000156005,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_DEVICE_GROUP_RENDER_PASS_BEGIN_INFO = 1000060003,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_POINT_CLIPPING_PROPERTIES = 1000117000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_RENDER_PASS_INPUT_ATTACHMENT_ASPECT_CREATE_INFO = 1000117001,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_DOMAIN_ORIGIN_STATE_CREATE_INFO =
1000117003,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_RENDER_PASS_MULTIVIEW_CREATE_INFO = 1000053000,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_FEATURES = 1000053001,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PROPERTIES = 1000053002,
// Provided by VK_VERSION_1_1
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DRAW_PARAMETERS_FEATURES = 1000063000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_1_FEATURES = 49,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_1_PROPERTIES = 50,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_2_FEATURES = 51,

```

```

// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_2_PROPERTIES = 52,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO = 1000147000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRIVER_PROPERTIES = 1000196000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_MEMORY_MODEL_FEATURES = 1000211000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_QUERY_RESET_FEATURES = 1000261000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_FEATURES = 1000207000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_PROPERTIES = 1000207001,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO = 1000207002,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_TIMELINE_SEMAPHORE_SUBMIT_INFO = 1000207003,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_SEMAPHORE_WAIT_INFO = 1000207004,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_SEMAPHORE_SIGNAL_INFO = 1000207005,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES = 1000257000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_INFO = 1000244001,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_BUFFER_OPAQUE_CAPTURE_ADDRESS_CREATE_INFO = 1000257002,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_MEMORY_OPAQUE_CAPTURE_ADDRESS_ALLOCATE_INFO = 1000257003,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_DEVICE_MEMORY_OPAQUE_CAPTURE_ADDRESS_INFO = 1000257004,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_8BIT_STORAGE_FEATURES = 1000177000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_INT64_FEATURES = 1000180000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_FLOAT16_INT8_FEATURES = 1000082000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FLOAT_CONTROLS_PROPERTIES = 1000197000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_BINDING_FLAGS_CREATE_INFO = 1000161000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_FEATURES = 1000161001,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_PROPERTIES = 1000161002,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_ALLOCATE_INFO =
1000161003,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_LAYOUT_SUPPORT =

```



```

1000161004,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCALAR_BLOCK_LAYOUT_FEATURES = 1000221000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_FILTER_MINMAX_PROPERTIES = 1000130000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_SAMPLER_REDUCTION_MODE_CREATE_INFO = 1000130001,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_UNIFORM_BUFFER_STANDARD_LAYOUT_FEATURES =
1000253000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_EXTENDED_TYPES_FEATURES =
1000175000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2 = 1000109000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2 = 1000109001,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_2 = 1000109002,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_SUBPASS_DEPENDENCY_2 = 1000109003,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO_2 = 1000109004,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_SUBPASS_BEGIN_INFO = 1000109005,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_SUBPASS_END_INFO = 1000109006,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_STENCIL_RESOLVE_PROPERTIES = 1000199000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_DEPTH_STENCIL_RESOLVE = 1000199001,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_IMAGE_STENCIL_USAGE_CREATE_INFO = 1000246000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGELESS_FRAMEBUFFER_FEATURES = 1000108000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENTS_CREATE_INFO = 1000108001,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENT_IMAGE_INFO = 1000108002,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_RENDER_PASS_ATTACHMENT_BEGIN_INFO = 1000108003,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SEPARATE_DEPTH_STENCIL_LAYOUTS_FEATURES =
1000241000,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_STENCIL_LAYOUT = 1000241001,
// Provided by VK_VERSION_1_2
VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_STENCIL_LAYOUT = 1000241002,
VK_STRUCTURE_TYPE_MEMORY_BARRIER_2 = 1000314000,
VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER_2 = 1000314001,
VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER_2 = 1000314002,

```

```

VK_STRUCTURE_TYPE_DEPENDENCY_INFO = 1000314003,
VK_STRUCTURE_TYPE_SUBMIT_INFO_2 = 1000314004,
VK_STRUCTURE_TYPE_SEMAPHORE_SUBMIT_INFO = 1000314005,
VK_STRUCTURE_TYPE_COMMAND_BUFFER_SUBMIT_INFO = 1000314006,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SYNCHRONIZATION_2_FEATURES = 1000314007,
VK_STRUCTURE_TYPE_COPY_BUFFER_INFO_2 = 1000337000,
VK_STRUCTURE_TYPE_COPY_IMAGE_INFO_2 = 1000337001,
VK_STRUCTURE_TYPE_COPY_BUFFER_TO_IMAGE_INFO_2 = 1000337002,
VK_STRUCTURE_TYPE_COPY_IMAGE_TO_BUFFER_INFO_2 = 1000337003,
VK_STRUCTURE_TYPE_BUFFER_COPY_2 = 1000337006,
VK_STRUCTURE_TYPE_IMAGE_COPY_2 = 1000337007,
VK_STRUCTURE_TYPE_BUFFER_IMAGE_COPY_2 = 1000337009,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXTURE_COMPRESSION_ASTC_HDR_FEATURES =
1000066000,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TERMINATE_INVOCATION_FEATURES =
1000215000,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DEMOTE_TO_HELPER_INVOCATION_FEATURES =
1000276000,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ROBUSTNESS_FEATURES = 1000335000,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_PROPERTIES = 1000225000,
VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_REQUIRED_SUBGROUP_SIZE_CREATE_INFO =
1000225001,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_FEATURES = 1000225002,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_PROPERTIES = 1000281001,
VK_STRUCTURE_TYPE_BLIT_IMAGE_INFO_2 = 1000337004,
VK_STRUCTURE_TYPE_RESOLVE_IMAGE_INFO_2 = 1000337005,
VK_STRUCTURE_TYPE_IMAGE_BLIT_2 = 1000337008,
VK_STRUCTURE_TYPE_IMAGE_RESOLVE_2 = 1000337010,
VK_STRUCTURE_TYPE_DEVICE_QUEUE_GLOBAL_PRIORITY_CREATE_INFO = 1000174000,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GLOBAL_PRIORITY_QUERY_FEATURES = 1000388000,
VK_STRUCTURE_TYPE_QUEUE_FAMILY_GLOBAL_PRIORITY_PROPERTIES = 1000388001,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES = 1000265000,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES = 1000259000,
VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO = 1000259001,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES = 1000259002,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_PROPERTIES =
1000525000,
VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_DIVISOR_STATE_CREATE_INFO = 1000190001,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_FEATURES = 1000190002,
// Provided by VKSC_VERSION_1_0
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_SC_1_0_FEATURES = 1000298000,
// Provided by VKSC_VERSION_1_0
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_SC_1_0_PROPERTIES = 1000298001,
// Provided by VKSC_VERSION_1_0
VK_STRUCTURE_TYPE_DEVICE_OBJECT_RESERVATION_CREATE_INFO = 1000298002,
// Provided by VKSC_VERSION_1_0
VK_STRUCTURE_TYPE_COMMAND_POOL_MEMORY_RESERVATION_CREATE_INFO = 1000298003,
// Provided by VKSC_VERSION_1_0
VK_STRUCTURE_TYPE_COMMAND_POOL_MEMORY_CONSUMPTION = 1000298004,
// Provided by VKSC_VERSION_1_0
VK_STRUCTURE_TYPE_PIPELINE_POOL_SIZE = 1000298005,

```

```

// Provided by VKSC_VERSION_1_0
VK_STRUCTURE_TYPE_FAULT_DATA = 1000298007,
// Provided by VKSC_VERSION_1_0
VK_STRUCTURE_TYPE_FAULT_CALLBACK_INFO = 1000298008,
// Provided by VKSC_VERSION_1_0
VK_STRUCTURE_TYPE_PIPELINE_OFFLINE_CREATE_INFO = 1000298010,
// Provided by VK_KHR_swapchain
VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR = 1000001000,
// Provided by VK_KHR_swapchain
VK_STRUCTURE_TYPE_PRESENT_INFO_KHR = 1000001001,
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_CAPABILITIES_KHR = 1000060007,
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
VK_STRUCTURE_TYPE_IMAGE_SWAPCHAIN_CREATE_INFO_KHR = 1000060008,
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_SWAPCHAIN_INFO_KHR = 1000060009,
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
VK_STRUCTURE_TYPE_ACQUIRE_NEXT_IMAGE_INFO_KHR = 1000060010,
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_INFO_KHR = 1000060011,
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
VK_STRUCTURE_TYPE_DEVICE_GROUP_SWAPCHAIN_CREATE_INFO_KHR = 1000060012,
// Provided by VK_KHR_display
VK_STRUCTURE_TYPE_DISPLAY_MODE_CREATE_INFO_KHR = 1000002000,
// Provided by VK_KHR_display
VK_STRUCTURE_TYPE_DISPLAY_SURFACE_CREATE_INFO_KHR = 1000002001,
// Provided by VK_KHR_display_swapchain
VK_STRUCTURE_TYPE_DISPLAY_PRESENT_INFO_KHR = 1000003000,
// Provided by VK_NV_private_vendor_info
VK_STRUCTURE_TYPE_PRIVATE_VENDOR_INFO_PLACEHOLDER_OFFSET_0_NV = 1000051000,
// Provided by VK_EXT_astc_decode_mode
VK_STRUCTURE_TYPE_IMAGE_VIEW_ASTC_DECODE_MODE_EXT = 1000067000,
// Provided by VK_EXT_astc_decode_mode
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ASTC_DECODE_FEATURES_EXT = 1000067001,
// Provided by VK_KHR_external_memory_fd
VK_STRUCTURE_TYPE_IMPORT_MEMORY_FD_INFO_KHR = 1000074000,
// Provided by VK_KHR_external_memory_fd
VK_STRUCTURE_TYPE_MEMORY_FD_PROPERTIES_KHR = 1000074001,
// Provided by VK_KHR_external_memory_fd
VK_STRUCTURE_TYPE_MEMORY_GET_FD_INFO_KHR = 1000074002,
// Provided by VK_KHR_external_semaphore_fd
VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_FD_INFO_KHR = 1000079000,
// Provided by VK_KHR_external_semaphore_fd
VK_STRUCTURE_TYPE_SEMAPHORE_GET_FD_INFO_KHR = 1000079001,
// Provided by VK_KHR_incremental_present
VK_STRUCTURE_TYPE_PRESENT_REGIONS_KHR = 1000084000,
// Provided by VK_EXT_display_surface_counter
VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_EXT = 1000090000,
// Provided by VK_EXT_display_control
VK_STRUCTURE_TYPE_DISPLAY_POWER_INFO_EXT = 1000091000,
// Provided by VK_EXT_display_control

```

```

VK_STRUCTURE_TYPE_DEVICE_EVENT_INFO_EXT = 1000091001,
// Provided by VK_EXT_display_control
VK_STRUCTURE_TYPE_DISPLAY_EVENT_INFO_EXT = 1000091002,
// Provided by VK_EXT_display_control
VK_STRUCTURE_TYPE_SWAPCHAIN_COUNTER_CREATE_INFO_EXT = 1000091003,
// Provided by VK_EXT_discard_rectangles
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DISCARD_RECTANGLE_PROPERTIES_EXT = 1000099000,
// Provided by VK_EXT_discard_rectangles
VK_STRUCTURE_TYPE_PIPELINE_DISCARD_RECTANGLE_STATE_CREATE_INFO_EXT = 1000099001,
// Provided by VK_EXT_conservative_rasterization
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CONSERVATIVE_RASTERIZATION_PROPERTIES_EXT =
1000101000,
// Provided by VK_EXT_conservative_rasterization
VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_CONSERVATIVE_STATE_CREATE_INFO_EXT =
1000101001,
// Provided by VK_EXT_depth_clip_enable
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_CLIP_ENABLE_FEATURES_EXT = 1000102000,
// Provided by VK_EXT_depth_clip_enable
VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_DEPTH_CLIP_STATE_CREATE_INFO_EXT =
1000102001,
// Provided by VK_EXT_hdr_metadata
VK_STRUCTURE_TYPE_HDR_METADATA_EXT = 1000105000,
// Provided by VK_KHR_shared_presentable_image
VK_STRUCTURE_TYPE_SHARED_PRESENT_SURFACE_CAPABILITIES_KHR = 1000111000,
// Provided by VK_KHR_external_fence_fd
VK_STRUCTURE_TYPE_IMPORT_FENCE_FD_INFO_KHR = 1000115000,
// Provided by VK_KHR_external_fence_fd
VK_STRUCTURE_TYPE_FENCE_GET_FD_INFO_KHR = 1000115001,
// Provided by VK_KHR_performance_query
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PERFORMANCE_QUERY_FEATURES_KHR = 1000116000,
// Provided by VK_KHR_performance_query
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PERFORMANCE_QUERY_PROPERTIES_KHR = 1000116001,
// Provided by VK_KHR_performance_query
VK_STRUCTURE_TYPE_QUERY_POOL_PERFORMANCE_CREATE_INFO_KHR = 1000116002,
// Provided by VK_KHR_performance_query
VK_STRUCTURE_TYPE_PERFORMANCE_QUERY_SUBMIT_INFO_KHR = 1000116003,
// Provided by VK_KHR_performance_query
VK_STRUCTURE_TYPE_ACQUIRE_PROFILING_LOCK_INFO_KHR = 1000116004,
// Provided by VK_KHR_performance_query
VK_STRUCTURE_TYPE_PERFORMANCE_COUNTER_KHR = 1000116005,
// Provided by VK_KHR_performance_query
VK_STRUCTURE_TYPE_PERFORMANCE_COUNTER_DESCRIPTION_KHR = 1000116006,
// Provided by VKSC_VERSION_1_0 with VK_KHR_performance_query
VK_STRUCTURE_TYPE_PERFORMANCE_QUERY_RESERVATION_INFO_KHR = 1000116007,
// Provided by VK_KHR_get_surface_capabilities2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SURFACE_INFO_2_KHR = 1000119000,
// Provided by VK_KHR_get_surface_capabilities2
VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_KHR = 1000119001,
// Provided by VK_KHR_get_surface_capabilities2
VK_STRUCTURE_TYPE_SURFACE_FORMAT_2_KHR = 1000119002,
// Provided by VK_KHR_get_display_properties2

```

```

VK_STRUCTURE_TYPE_DISPLAY_PROPERTIES_2_KHR = 1000121000,
// Provided by VK_KHR_get_display_properties2
VK_STRUCTURE_TYPE_DISPLAY_PLANE_PROPERTIES_2_KHR = 1000121001,
// Provided by VK_KHR_get_display_properties2
VK_STRUCTURE_TYPE_DISPLAY_MODE_PROPERTIES_2_KHR = 1000121002,
// Provided by VK_KHR_get_display_properties2
VK_STRUCTURE_TYPE_DISPLAY_PLANE_INFO_2_KHR = 1000121003,
// Provided by VK_KHR_get_display_properties2
VK_STRUCTURE_TYPE_DISPLAY_PLANE_CAPABILITIES_2_KHR = 1000121004,
// Provided by VK_EXT_debug_utils
VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT = 1000128000,
// Provided by VK_EXT_debug_utils
VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_TAG_INFO_EXT = 1000128001,
// Provided by VK_EXT_debug_utils
VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT = 1000128002,
// Provided by VK_EXT_debug_utils
VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CALLBACK_DATA_EXT = 1000128003,
// Provided by VK_EXT_debug_utils
VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT = 1000128004,
// Provided by VK_EXT_sample_locations
VK_STRUCTURE_TYPE_SAMPLE_LOCATIONS_INFO_EXT = 1000143000,
// Provided by VK_EXT_sample_locations
VK_STRUCTURE_TYPE_RENDER_PASS_SAMPLE_LOCATIONS_BEGIN_INFO_EXT = 1000143001,
// Provided by VK_EXT_sample_locations
VK_STRUCTURE_TYPE_PIPELINE_SAMPLE_LOCATIONS_STATE_CREATE_INFO_EXT = 1000143002,
// Provided by VK_EXT_sample_locations
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLE_LOCATIONS_PROPERTIES_EXT = 1000143003,
// Provided by VK_EXT_sample_locations
VK_STRUCTURE_TYPE_MULTISAMPLE_PROPERTIES_EXT = 1000143004,
// Provided by VK_EXT_blend_operation_advanced
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BLEND_OPERATION_ADVANCED_FEATURES_EXT =
1000148000,
// Provided by VK_EXT_blend_operation_advanced
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BLEND_OPERATION_ADVANCED_PROPERTIES_EXT =
1000148001,
// Provided by VK_EXT_blend_operation_advanced
VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_ADVANCED_STATE_CREATE_INFO_EXT =
1000148002,
// Provided by VK_EXT_image_drm_format_modifier
VK_STRUCTURE_TYPE_DRM_FORMAT_MODIFIER_PROPERTIES_LIST_EXT = 1000158000,
// Provided by VK_EXT_image_drm_format_modifier
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_DRM_FORMAT_MODIFIER_INFO_EXT = 1000158002,
// Provided by VK_EXT_image_drm_format_modifier
VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_LIST_CREATE_INFO_EXT = 1000158003,
// Provided by VK_EXT_image_drm_format_modifier
VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_EXPLICIT_CREATE_INFO_EXT = 1000158004,
// Provided by VK_EXT_image_drm_format_modifier
VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_PROPERTIES_EXT = 1000158005,
// Provided by VK_EXT_image_drm_format_modifier with VK_KHR_format_feature_flags2 or
VK_VERSION_1_3
VK_STRUCTURE_TYPE_DRM_FORMAT_MODIFIER_PROPERTIES_LIST_2_EXT = 1000158006,

```

```

// Provided by VK_EXT_filter_cubic
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_VIEW_IMAGE_FORMAT_INFO_EXT = 1000170000,
// Provided by VK_EXT_filter_cubic
VK_STRUCTURE_TYPE_FILTER_CUBIC_IMAGE_VIEW_IMAGE_FORMAT_PROPERTIES_EXT =
1000170001,
// Provided by VK_EXT_external_memory_host
VK_STRUCTURE_TYPE_IMPORT_MEMORY_HOST_POINTER_INFO_EXT = 1000178000,
// Provided by VK_EXT_external_memory_host
VK_STRUCTURE_TYPE_MEMORY_HOST_POINTER_PROPERTIES_EXT = 1000178001,
// Provided by VK_EXT_external_memory_host
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_HOST_PROPERTIES_EXT =
1000178002,
// Provided by VK_KHR_shader_clock
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CLOCK_FEATURES_KHR = 1000181000,
// Provided by VK_EXT_pci_bus_info
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PCI_BUS_INFO_PROPERTIES_EXT = 1000212000,
// Provided by VK_KHR_fragment_shading_rate
VK_STRUCTURE_TYPE_FRAGMENT_SHADING_RATE_ATTACHMENT_INFO_KHR = 1000226000,
// Provided by VK_KHR_fragment_shading_rate
VK_STRUCTURE_TYPE_PIPELINE_FRAGMENT_SHADING_RATE_STATE_CREATE_INFO_KHR =
1000226001,
// Provided by VK_KHR_fragment_shading_rate
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_PROPERTIES_KHR =
1000226002,
// Provided by VK_KHR_fragment_shading_rate
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_FEATURES_KHR = 1000226003,
// Provided by VK_KHR_fragment_shading_rate
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_KHR = 1000226004,
// Provided by VK_KHR_fragment_shading_rate with VK_VERSION_1_3 or
VK_KHR_dynamic_rendering
VK_STRUCTURE_TYPE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_INFO_KHR =
1000044006,
// Provided by VK_EXT_shader_image_atomic_int64
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_IMAGE_ATOMIC_INT64_FEATURES_EXT =
1000234000,
// Provided by VK_EXT_memory_budget
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_BUDGET_PROPERTIES_EXT = 1000237000,
// Provided by VK_EXT_validation_features
VK_STRUCTURE_TYPE_VALIDATION_FEATURES_EXT = 1000247000,
// Provided by VK_EXT_fragment_shader_interlock
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_INTERLOCK_FEATURES_EXT =
1000251000,
// Provided by VK_EXT_ycbcr_image_arrays
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_IMAGE_ARRAYS_FEATURES_EXT = 1000252000,
// Provided by VK_EXT_headless_surface
VK_STRUCTURE_TYPE_HEADLESS_SURFACE_CREATE_INFO_EXT = 1000256000,
// Provided by VK_EXT_shader_atomic_float
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_FLOAT_FEATURES_EXT = 1000260000,
// Provided by VK_EXT_extended_dynamic_state
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_FEATURES_EXT =
1000267000,

```



```

// Provided by VK_EXT_texel_buffer_alignment
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_FEATURES_EXT =
1000281000,
// Provided by VK_EXT_custom_border_color
VK_STRUCTURE_TYPE_SAMPLER_CUSTOM_BORDER_COLOR_CREATE_INFO_EXT = 1000287000,
// Provided by VK_EXT_custom_border_color
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUSTOM_BORDER_COLOR_PROPERTIES_EXT = 1000287001,
// Provided by VK_EXT_custom_border_color
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUSTOM_BORDER_COLOR_FEATURES_EXT = 1000287002,
// Provided by VK_KHR_object_refresh
VK_STRUCTURE_TYPE_REFRESH_OBJECT_LIST_KHR = 1000308000,
// Provided by VK_EXT_ycbcr_2plane_444_formats
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_2_PLANE_444_FORMATS_FEATURES_EXT =
1000330000,
// Provided by VK_EXT_4444_formats
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_4444_FORMATS_FEATURES_EXT = 1000340000,
// Provided by VK_EXT_vertex_input_dynamic_state
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_INPUT_DYNAMIC_STATE_FEATURES_EXT =
1000352000,
// Provided by VK_EXT_vertex_input_dynamic_state
VK_STRUCTURE_TYPE_VERTEX_INPUT_BINDING_DESCRIPTION_2_EXT = 1000352001,
// Provided by VK_EXT_vertex_input_dynamic_state
VK_STRUCTURE_TYPE_VERTEX_INPUT_ATTRIBUTE_DESCRIPTION_2_EXT = 1000352002,
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
VK_STRUCTURE_TYPE_IMPORT_FENCE_SCI_SYNC_INFO_NV = 1000373000,
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
VK_STRUCTURE_TYPE_EXPORT_FENCE_SCI_SYNC_INFO_NV = 1000373001,
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
VK_STRUCTURE_TYPE_FENCE_GET_SCI_SYNC_INFO_NV = 1000373002,
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
VK_STRUCTURE_TYPE_SCI_SYNC_ATTRIBUTES_INFO_NV = 1000373003,
// Provided by VK_NV_external_sci_sync
VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_SCI_SYNC_INFO_NV = 1000373004,
// Provided by VK_NV_external_sci_sync
VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_SCI_SYNC_INFO_NV = 1000373005,
// Provided by VK_NV_external_sci_sync
VK_STRUCTURE_TYPE_SEMAPHORE_GET_SCI_SYNC_INFO_NV = 1000373006,
// Provided by VK_NV_external_sci_sync
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SCI_SYNC_FEATURES_NV = 1000373007,
// Provided by VK_NV_external_memory_sci_buf
VK_STRUCTURE_TYPE_IMPORT_MEMORY_SCI_BUF_INFO_NV = 1000374000,
// Provided by VK_NV_external_memory_sci_buf
VK_STRUCTURE_TYPE_EXPORT_MEMORY_SCI_BUF_INFO_NV = 1000374001,
// Provided by VK_NV_external_memory_sci_buf
VK_STRUCTURE_TYPE_MEMORY_GET_SCI_BUF_INFO_NV = 1000374002,
// Provided by VK_NV_external_memory_sci_buf
VK_STRUCTURE_TYPE_MEMORY_SCI_BUF_PROPERTIES_NV = 1000374003,
// Provided by VK_NV_external_memory_sci_buf
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_SCI_BUF_FEATURES_NV =
1000374004,
// Provided by VK_EXT_extended_dynamic_state2

```

```

VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_2_FEATURES_EXT =
1000377000,
// Provided by VK_EXT_color_write_enable
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COLOR_WRITE_ENABLE_FEATURES_EXT = 1000381000,
// Provided by VK_EXT_color_write_enable
VK_STRUCTURE_TYPE_PIPELINE_COLOR_WRITE_CREATE_INFO_EXT = 1000381001,
// Provided by VK_EXT_application_parameters
VK_STRUCTURE_TYPE_APPLICATION_PARAMETERS_EXT = 1000435000,
// Provided by VK_NV_external_sci_sync2
VK_STRUCTURE_TYPE_SEMAPHORE_SCI_SYNC_POOL_CREATE_INFO_NV = 1000489000,
// Provided by VK_NV_external_sci_sync2
VK_STRUCTURE_TYPE_SEMAPHORE_SCI_SYNC_CREATE_INFO_NV = 1000489001,
// Provided by VK_NV_external_sci_sync2
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SCI_SYNC_2_FEATURES_NV = 1000489002,
// Provided by VKSC_VERSION_1_0 with VK_NV_external_sci_sync2
VK_STRUCTURE_TYPE_DEVICE_SEMAPHORE_SCI_SYNC_POOL_RESERVATION_CREATE_INFO_NV =
1000489003,
// Provided by VK_EXT_layer_settings
VK_STRUCTURE_TYPE_LAYER_SETTINGS_CREATE_INFO_EXT = 1000496000,
// Provided by VK_QNX_external_memory_screen_buffer
VK_STRUCTURE_TYPE_SCREEN_BUFFER_PROPERTIES_QNX = 1000529000,
// Provided by VK_QNX_external_memory_screen_buffer
VK_STRUCTURE_TYPE_SCREEN_BUFFER_FORMAT_PROPERTIES_QNX = 1000529001,
// Provided by VK_QNX_external_memory_screen_buffer
VK_STRUCTURE_TYPE_IMPORT_SCREEN_BUFFER_INFO_QNX = 1000529002,
// Provided by VK_QNX_external_memory_screen_buffer
VK_STRUCTURE_TYPE_EXTERNAL_FORMAT_QNX = 1000529003,
// Provided by VK_QNX_external_memory_screen_buffer
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_SCREEN_BUFFER_FEATURES_QNX =
1000529004,
// Provided by VK_KHR_calibrated_timestamps
VK_STRUCTURE_TYPE_CALIBRATED_TIMESTAMP_INFO_KHR = 1000184000,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_FEATURES_KHR = 1000286000,
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_PROPERTIES_KHR = 1000286001,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXTURE_COMPRESSION_ASTC_HDR_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXTURE_COMPRESSION_ASTC_HDR_FEATURES,
// Provided by VK_KHR_global_priority
VK_STRUCTURE_TYPE_DEVICE_QUEUE_GLOBAL_PRIORITY_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_DEVICE_QUEUE_GLOBAL_PRIORITY_CREATE_INFO,
// Provided by VK_KHR_global_priority
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GLOBAL_PRIORITY_QUERY_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GLOBAL_PRIORITY_QUERY_FEATURES,
// Provided by VK_KHR_global_priority
VK_STRUCTURE_TYPE_QUEUE_FAMILY_GLOBAL_PRIORITY_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_QUEUE_FAMILY_GLOBAL_PRIORITY_PROPERTIES,
// Provided by VK_KHR_shader_terminate_invocation
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TERMINATE_INVOCATION_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TERMINATE_INVOCATION_FEATURES,
// Provided by VK_EXT_subgroup_size_control
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_PROPERTIES_EXT =

```



```

VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_PROPERTIES,
    // Provided by VK_EXT_subgroup_size_control
    VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_REQUIRED_SUBGROUP_SIZE_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_REQUIRED_SUBGROUP_SIZE_CREATE_INFO,
    // Provided by VK_EXT_subgroup_size_control
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_FEATURES,
    // Provided by VK_EXT_line_rasterization
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES,
    // Provided by VK_EXT_line_rasterization
    VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO_EXT =
VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO,
    // Provided by VK_EXT_line_rasterization
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES,
    // Provided by VK_EXT_index_type_uint8
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES,
    // Provided by VK_EXT_shader_demote_to_helper_invocation
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DEMOTE_TO_HELPER_INVOCATION_FEATURES_EXT
= VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DEMOTE_TO_HELPER_INVOCATION_FEATURES,
    // Provided by VK_EXT_texel_buffer_alignment
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_PROPERTIES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_PROPERTIES,
    // Provided by VK_EXT_robustness2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_FEATURES_KHR,
    // Provided by VK_EXT_robustness2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_PROPERTIES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_PROPERTIES_KHR,
    // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_MEMORY_BARRIER_2_KHR = VK_STRUCTURE_TYPE_MEMORY_BARRIER_2,
    // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER_2_KHR =
VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER_2,
    // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER_2_KHR =
VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER_2,
    // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_DEPENDENCY_INFO_KHR = VK_STRUCTURE_TYPE_DEPENDENCY_INFO,
    // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_SUBMIT_INFO_2_KHR = VK_STRUCTURE_TYPE_SUBMIT_INFO_2,
    // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_SEMAPHORE_SUBMIT_INFO_KHR =
VK_STRUCTURE_TYPE_SEMAPHORE_SUBMIT_INFO,
    // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_COMMAND_BUFFER_SUBMIT_INFO_KHR =
VK_STRUCTURE_TYPE_COMMAND_BUFFER_SUBMIT_INFO,
    // Provided by VK_KHR_synchronization2
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SYNCHRONIZATION_2_FEATURES_KHR =

```

```

VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SYNCHRONIZATION_2_FEATURES,
    // Provided by VK_EXT_image_robustness
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ROBUSTNESS_FEATURES_EXT =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ROBUSTNESS_FEATURES,
    // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_COPY_BUFFER_INFO_2_KHR = VK_STRUCTURE_TYPE_COPY_BUFFER_INFO_2,
    // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_COPY_IMAGE_INFO_2_KHR = VK_STRUCTURE_TYPE_COPY_IMAGE_INFO_2,
    // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_COPY_BUFFER_TO_IMAGE_INFO_2_KHR =
VK_STRUCTURE_TYPE_COPY_BUFFER_TO_IMAGE_INFO_2,
    // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_COPY_IMAGE_TO_BUFFER_INFO_2_KHR =
VK_STRUCTURE_TYPE_COPY_IMAGE_TO_BUFFER_INFO_2,
    // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_BLIT_IMAGE_INFO_2_KHR = VK_STRUCTURE_TYPE_BLIT_IMAGE_INFO_2,
    // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_RESOLVE_IMAGE_INFO_2_KHR =
VK_STRUCTURE_TYPE_RESOLVE_IMAGE_INFO_2,
    // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_BUFFER_COPY_2_KHR = VK_STRUCTURE_TYPE_BUFFER_COPY_2,
    // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_IMAGE_COPY_2_KHR = VK_STRUCTURE_TYPE_IMAGE_COPY_2,
    // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_IMAGE_BLIT_2_KHR = VK_STRUCTURE_TYPE_IMAGE_BLIT_2,
    // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_BUFFER_IMAGE_COPY_2_KHR = VK_STRUCTURE_TYPE_BUFFER_IMAGE_COPY_2,
    // Provided by VK_KHR_copy_commands2
    VK_STRUCTURE_TYPE_IMAGE_RESOLVE_2_KHR = VK_STRUCTURE_TYPE_IMAGE_RESOLVE_2,
    // Provided by VK_NV_external_memory_sci_buf
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SCI_BUF_FEATURES_NV =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_SCI_BUF_FEATURES_NV,
    // Provided by VK_KHR_vertex_attribute_divisor
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_PROPERTIES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_PROPERTIES,
    // Provided by VK_KHR_vertex_attribute_divisor
    VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_DIVISOR_STATE_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_DIVISOR_STATE_CREATE_INFO,
    // Provided by VK_KHR_vertex_attribute_divisor
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_FEATURES,
    // Provided by VK_KHR_index_type_uint8
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES,
    // Provided by VK_KHR_line_rasterization
    VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES_KHR =
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES,
    // Provided by VK_KHR_line_rasterization
    VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO_KHR =
VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO,
    // Provided by VK_KHR_line_rasterization

```

```
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES_KHR =  
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES,  
} VkStructureType;
```

3.13. API Name Aliases

A small number of APIs did not follow the [naming conventions](#) when initially defined. For consistency, when we discover an API name that violates the naming conventions, we rename it in the Specification, XML, and header files. For backwards compatibility, the original (incorrect) name is retained as a “typo alias”. The alias is legacy and should not be used, but will be retained indefinitely.

NOTE

`VK_STENCIL_FRONT_AND_BACK` is an example of a *typo alias*. It was initially defined as part of `VkStencilFaceFlagBits`. Once the naming inconsistency was noticed, it was renamed to `VK_STENCIL_FACE_FRONT_AND_BACK`, and the old name was aliased to the correct name.

Chapter 4. Initialization

Before using Vulkan, an application **must** initialize it by loading the Vulkan commands, and creating a `VkInstance` object.

4.1. Command Function Pointers

Vulkan commands are not necessarily exposed by static linking on a platform. Commands to query function pointers for Vulkan commands are described below.

NOTE

When extensions are [promoted](#) or otherwise incorporated into another extension or Vulkan core version, command [aliases](#) may be included. Whilst the behavior of each command alias is identical, the behavior of retrieving each alias's function pointer is not. A function pointer for a given alias can only be retrieved if the extension or version that introduced that alias is supported and enabled, irrespective of whether any other alias is available.

Function pointers for all Vulkan commands **can** be obtained by calling:

```
// Provided by VK_VERSION_1_0
PFN_vkVoidFunction vkGetInstanceProcAddr(
    VkInstance          instance,
    const char*         pName);
```

- `instance` is the instance that the function pointer will be compatible with, or `NULL` for commands not dependent on any instance.
- `pName` is the name of the command to obtain.

`vkGetInstanceProcAddr` itself is obtained in a platform- and loader- specific manner. Typically, the loader library will export this command as a function symbol, so applications **can** link against the loader library, or load it dynamically and look up the symbol using platform-specific APIs.

The table below defines the various use cases for `vkGetInstanceProcAddr` and expected return value (“fp” is “function pointer”) for each case. A valid returned function pointer (“fp”) **must** not be `NULL`.

The returned function pointer is of type `PFN_vkVoidFunction`, and **must** be cast to the type of the command being queried before use.

Table 2. `vkGetInstanceProcAddr` behavior

<code>instance</code>	<code>pName</code>	return value
* ¹	<code>NULL</code>	undefined
invalid non- <code>NULL</code> instance	* ¹	undefined
<code>NULL</code>	<i>global command</i> ²	fp
<code>NULL</code>	<code>vkGetInstanceProcAddr</code>	fp ⁵

instance	pName	return value
instance	vkGetInstanceProcAddr	fp
instance	core <i>dispatchable command</i>	fp ³
instance	enabled instance extension dispatchable command for instance	fp ³
instance	available device extension ⁴ dispatchable command for instance	fp ³
any other case, not covered above		NULL

1

"*" means any representable value for the parameter (including valid values, invalid values, and **NULL**).

2

The global commands are: [vkEnumerateInstanceVersion](#), [vkEnumerateInstanceExtensionProperties](#), [vkEnumerateInstanceLayerProperties](#), and [vkCreateInstance](#). Dispatchable commands are all other commands which are not global.

3

The returned function pointer **must** only be called with a dispatchable object (the first parameter) that is **instance** or a child of **instance**, e.g. [VkInstance](#), [VkPhysicalDevice](#), [VkDevice](#), [VkQueue](#), or [VkCommandBuffer](#).

4

An “available device extension” is a device extension supported by any physical device enumerated by **instance**.

5

[vkGetInstanceProcAddr](#) can resolve itself with a **NULL** instance pointer.

Valid Usage (Implicit)

- VUID-vkGetInstanceProcAddr-instance-parameter
If **instance** is not **NULL**, **instance** **must** be a valid [VkInstance](#) handle
- VUID-vkGetInstanceProcAddr-pName-parameter
pName **must** be a null-terminated UTF-8 string

In order to support systems with multiple Vulkan implementations, the function pointers returned by [vkGetInstanceProcAddr](#) **may** point to dispatch code that calls a different real implementation for different [VkDevice](#) objects or their child objects. The overhead of the internal dispatch for [VkDevice](#) objects can be avoided by obtaining device-specific function pointers for any commands that use a

device or device-child object as their dispatchable object. Such function pointers **can** be obtained by calling:

```
// Provided by VK_VERSION_1_0
PFN_vkVoidFunction vkGetDeviceProcAddr(
    VkDevice          device,
    const char*       pName);
```

The table below defines the various use cases for `vkGetDeviceProcAddr` and expected return value (“fp” is “function pointer”) for each case. A valid returned function pointer (“fp”) **must** not be `NULL`.

The returned function pointer is of type `PFN_vkVoidFunction`, and **must** be cast to the type of the command being queried before use. The function pointer **must** only be called with a dispatchable object (the first parameter) that is `device` or a child of `device`.

Table 3. `vkGetDeviceProcAddr` behavior

device	pName	return value
<code>NULL</code>	* ¹	undefined
invalid device	* ¹	undefined
device	<code>NULL</code>	undefined
device	requested core version ² device-level dispatchable command ³	fp ⁴
device	enabled extension device-level dispatchable command ³	fp ⁴
any other case, not covered above		<code>NULL</code>

1
“*” means any representable value for the parameter (including valid values, invalid values, and `NULL`).

2
Device-level commands which are part of the core version specified by `VkApplicationInfo::apiVersion` when creating the instance will always return a valid function pointer. Core commands beyond that version which are supported by the implementation **may** either return `NULL` or a function pointer. If a function pointer is returned, it **must** not be called.

3
In this function, device-level excludes all physical-device-level commands.

4
The returned function pointer **must** only be called with a dispatchable object (the first parameter) that is `device` or a child of `device` e.g. `VkDevice`, `VkQueue`, or `VkCommandBuffer`.

Valid Usage (Implicit)

- VUID-vkGetDeviceProcAddr-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkGetDeviceProcAddr-pName-parameter
pName **must** be a null-terminated UTF-8 string

The definition of [PFN_vkVoidFunction](#) is:

```
// Provided by VK_VERSION_1_0
typedef void (VKAPI_PTR *PFN_vkVoidFunction)(void);
```

This type is returned from command function pointer queries, and **must** be cast to an actual command function pointer before use.

4.1.1. Extending Physical Device Core Functionality

New core physical-device-level functionality **can** be used when both [VkPhysicalDeviceProperties::apiVersion](#) and [VkApplicationInfo::apiVersion](#) are greater than or equal to the version of Vulkan that added the new functionality. The Vulkan version supported by a physical device **can** be obtained by calling [vkGetPhysicalDeviceProperties](#).

4.1.2. Extending Physical Device From Device Extensions

In Vulkan SC 1.0, physical-device-level functionality of a device extension **can** be used with a physical device if the corresponding extension is enumerated by [vkEnumerateDeviceExtensionProperties](#) for that physical device, even before a logical device has been created.

To obtain a function pointer for a physical-device-level command from a device extension, an application **can** use [vkGetInstanceProcAddr](#). This function pointer **may** point to dispatch code, which calls a different real implementation for different [VkPhysicalDevice](#) objects. Applications **must** not use a [VkPhysicalDevice](#) in any command added by an extension or core version that is not supported by that physical device.

Device extensions **may** define structures that **can** be added to the **pNext** chain of physical-device-level commands.

4.2. Instances

There is no global state in Vulkan and all per-application state is stored in a [VkInstance](#) object. Creating a [VkInstance](#) object initializes the Vulkan library and allows the application to pass information about itself to the implementation.

Instances are represented by [VkInstance](#) handles:


```
// Provided by VK_VERSION_1_0
VK_DEFINE_HANDLE(VkInstance)
```

To query the version of instance-level functionality supported by the implementation, call:

```
// Provided by VK_VERSION_1_1
VkResult vkEnumerateInstanceVersion(
    uint32_t*                               pApiVersion);
```

- `pApiVersion` is a pointer to a `uint32_t`, which is the version of Vulkan supported by instance-level functionality, encoded as described in [Version Numbers](#).

NOTE

The intended behavior of `vkEnumerateInstanceVersion` is that an implementation **should** not need to perform memory allocations and **should** unconditionally return `VK_SUCCESS`. The loader, and any enabled layers, **may** return `VK_ERROR_OUT_OF_HOST_MEMORY` in the case of a failed memory allocation.

Valid Usage (Implicit)

- VUID-vkEnumerateInstanceVersion-pApiVersion-parameter `pApiVersion` **must** be a valid pointer to a `uint32_t` value

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To create an instance object, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateInstance(
    const VkInstanceCreateInfo*   pCreateInfo,
    const VkAllocationCallbacks* pAllocator,
    VkInstance*                   pInstance);
```

- `pCreateInfo` is a pointer to a `VkInstanceCreateInfo` structure controlling creation of the instance.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.

- `pInstance` points a `VkInstance` handle in which the resulting instance is returned.

`vkCreateInstance` verifies that the requested layers exist. If not, `vkCreateInstance` will return `VK_ERROR_LAYER_NOT_PRESENT`. Next `vkCreateInstance` verifies that the requested extensions are supported (e.g. in the implementation or in any enabled instance layer) and if any requested extension is not supported, `vkCreateInstance` **must** return `VK_ERROR_EXTENSION_NOT_PRESENT`. After verifying and enabling the instance layers and extensions the `VkInstance` object is created and returned to the application. If a requested extension is only supported by a layer, both the layer and the extension need to be specified at `vkCreateInstance` time for the creation to succeed.

Valid Usage

- VUID-vkCreateInstance-ppEnabledExtensionNames-01388

All **required extensions** for each extension in the `VkInstanceCreateInfo::ppEnabledExtensionNames` list **must** also be present in that list

Valid Usage (Implicit)

- VUID-vkCreateInstance-pCreateInfo-parameter
`pCreateInfo` **must** be a valid pointer to a valid `VkInstanceCreateInfo` structure
- VUID-vkCreateInstance-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateInstance-pInstance-parameter
`pInstance` **must** be a valid pointer to a `VkInstance` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_EXTENSION_NOT_PRESENT`
- `VK_ERROR_INCOMPATIBLE_DRIVER`
- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_LAYER_NOT_PRESENT`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkInstanceCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkInstanceCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkInstanceCreateFlags flags;
    const VkApplicationInfo* pApplicationInfo;
    uint32_t             enabledLayerCount;
    const char* const*   ppEnabledLayerNames;
    uint32_t             enabledExtensionCount;
    const char* const*   ppEnabledExtensionNames;
} VkInstanceCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkInstanceCreateFlagBits` indicating the behavior of the instance.
- `pApplicationInfo` is `NULL` or a pointer to a `VkApplicationInfo` structure. If not `NULL`, this information helps implementations recognize behavior inherent to classes of applications. `VkApplicationInfo` is defined in detail below.
- `enabledLayerCount` is the number of global layers to enable.
- `ppEnabledLayerNames` is a pointer to an array of `enabledLayerCount` null-terminated UTF-8 strings containing the names of layers to enable for the created instance. The layers are loaded in the order they are listed in this array, with the first array element being the closest to the application, and the last array element being the closest to the driver. See the [Layers](#) section for further details.
- `enabledExtensionCount` is the number of global extensions to enable.
- `ppEnabledExtensionNames` is a pointer to an array of `enabledExtensionCount` null-terminated UTF-8 strings containing the names of extensions to enable.

To capture events that occur while creating or destroying an instance, an application **can** link a `VkDebugUtilsMessengerCreateInfoEXT` structure to the `pNext` chain of the `VkInstanceCreateInfo` structure passed to `vkCreateInstance`. This callback is only valid for the duration of the `vkCreateInstance` and the `vkDestroyInstance` call. Use `vkCreateDebugUtilsMessengerEXT` to create persistent callback objects.

Valid Usage

- VUID-VkInstanceCreateInfo-pNext-04926
If the `pNext` chain of `VkInstanceCreateInfo` includes a `VkDebugUtilsMessengerCreateInfoEXT` structure, the list of enabled extensions in `ppEnabledExtensionNames` **must** contain `VK_EXT_debug_utils`
- VUID-VkInstanceCreateInfo-pNext-10242
If the `pNext` chain of `VkInstanceCreateInfo` includes a `VkLayerSettingsCreateInfoEXT` structure, the list of enabled extensions in `ppEnabledExtensionNames` **must** contain `VK_EXT_layer_settings`

- VUID-VkInstanceCreateInfo-pNext-10243

If the `pNext` chain of `VkInstanceCreateInfo` includes a `VkValidationFeaturesEXT` structure, the list of enabled extensions in `ppEnabledExtensionNames` **must** contain `VK_EXT_validation_features`

Valid Usage (Implicit)

- VUID-VkInstanceCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO`
- VUID-VkInstanceCreateInfo-pNext-pNext
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkDebugUtilsMessengerCreateInfoEXT`, `VkLayerSettingsCreateInfoEXT`, or `VkValidationFeaturesEXT`
- VUID-VkInstanceCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique, with the exception of structures of type `VkDebugUtilsMessengerCreateInfoEXT` or `VkLayerSettingsCreateInfoEXT`
- VUID-VkInstanceCreateInfo-flags-zero-bitmask
`flags` **must** be 0
- VUID-VkInstanceCreateInfo-pApplicationInfo-parameter
If `pApplicationInfo` is not `NULL`, `pApplicationInfo` **must** be a valid pointer to a valid `VkApplicationInfo` structure
- VUID-VkInstanceCreateInfo-ppEnabledLayerNames-parameter
If `enabledLayerCount` is not 0, `ppEnabledLayerNames` **must** be a valid pointer to an array of `enabledLayerCount` null-terminated UTF-8 strings
- VUID-VkInstanceCreateInfo-ppEnabledExtensionNames-parameter
If `enabledExtensionCount` is not 0, `ppEnabledExtensionNames` **must** be a valid pointer to an array of `enabledExtensionCount` null-terminated UTF-8 strings

```
// Provided by VK_VERSION_1_0
typedef enum VkInstanceCreateFlagBits {
} VkInstanceCreateFlagBits;
```

NOTE

All bits for this type are defined by extensions, and none of those extensions are enabled in this build of the specification.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkInstanceCreateFlags;
```

`VkInstanceCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

When creating a Vulkan instance for which you wish to enable or disable specific validation

features, add a [VkValidationFeaturesEXT](#) structure to the `pNext` chain of the [VkInstanceCreateInfo](#) structure, specifying the features to be enabled or disabled.

```
// Provided by VK_EXT_validation_features
typedef struct VkValidationFeaturesEXT {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           enabledValidationFeatureCount;
    const VkValidationFeatureEnableEXT* pEnabledValidationFeatures;
    uint32_t           disabledValidationFeatureCount;
    const VkValidationFeatureDisableEXT* pDisabledValidationFeatures;
} VkValidationFeaturesEXT;
```

- `sType` is a [VkStructureType](#) value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `enabledValidationFeatureCount` is the number of features to enable.
- `pEnabledValidationFeatures` is a pointer to an array of [VkValidationFeatureEnableEXT](#) values specifying the validation features to be enabled.
- `disabledValidationFeatureCount` is the number of features to disable.
- `pDisabledValidationFeatures` is a pointer to an array of [VkValidationFeatureDisableEXT](#) values specifying the validation features to be disabled.

Valid Usage

- VUID-VkValidationFeaturesEXT-pEnabledValidationFeatures-02967

If the `pEnabledValidationFeatures` array contains `VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_RESERVE_BINDING_SLOT_EXT`, then it **must** also contain `VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_EXT` or `VK_VALIDATION_FEATURE_ENABLE_DEBUG_PRINTF_EXT`

Valid Usage (Implicit)

- VUID-VkValidationFeaturesEXT-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_VALIDATION_FEATURES_EXT`

- VUID-VkValidationFeaturesEXT-pEnabledValidationFeatures-parameter

If `enabledValidationFeatureCount` is not 0, `pEnabledValidationFeatures` **must** be a valid pointer to an array of `enabledValidationFeatureCount` valid [VkValidationFeatureEnableEXT](#) values

- VUID-VkValidationFeaturesEXT-pDisabledValidationFeatures-parameter

If `disabledValidationFeatureCount` is not 0, `pDisabledValidationFeatures` **must** be a valid pointer to an array of `disabledValidationFeatureCount` valid [VkValidationFeatureDisableEXT](#) values

Possible values of elements of the `VkValidationFeaturesEXT::pEnabledValidationFeatures` array, specifying validation features to be enabled, are:

```
// Provided by VK_EXT_validation_features
typedef enum VkValidationFeatureEnableEXT {
    VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_EXT = 0,
    VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_RESERVE_BINDING_SLOT_EXT = 1,
    VK_VALIDATION_FEATURE_ENABLE_BEST_PRACTICES_EXT = 2,
    VK_VALIDATION_FEATURE_ENABLE_DEBUG_PRINTF_EXT = 3,
    VK_VALIDATION_FEATURE_ENABLE_SYNCHRONIZATION_VALIDATION_EXT = 4,
} VkValidationFeatureEnableEXT;
```

- `VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_EXT` specifies that GPU-assisted validation is enabled. Activating this feature instruments shader programs to generate additional diagnostic data. This feature is disabled by default.
- `VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_RESERVE_BINDING_SLOT_EXT` specifies that the validation layers reserve a descriptor set binding slot for their own use. The layer reports a value for `VkPhysicalDeviceLimits::maxBoundDescriptorSets` that is one less than the value reported by the device. If the device supports the binding of only one descriptor set, the validation layer does not perform GPU-assisted validation. This feature is disabled by default.
- `VK_VALIDATION_FEATURE_ENABLE_BEST_PRACTICES_EXT` specifies that Vulkan best-practices validation is enabled. Activating this feature enables the output of warnings related to common misuse of the API, but which are not explicitly prohibited by the specification. This feature is disabled by default.
- `VK_VALIDATION_FEATURE_ENABLE_DEBUG_PRINTF_EXT` specifies that the layers will process `debugPrintfEXT` operations in shaders and send the resulting output to the debug callback. This feature is disabled by default.
- `VK_VALIDATION_FEATURE_ENABLE_SYNCHRONIZATION_VALIDATION_EXT` specifies that Vulkan synchronization validation is enabled. This feature reports resource access conflicts due to missing or incorrect synchronization operations between actions (Draw, Copy, Dispatch, Blit) reading or writing the same regions of memory. This feature is disabled by default.

Possible values of elements of the `VkValidationFeaturesEXT::pDisabledValidationFeatures` array, specifying validation features to be disabled, are:

```
// Provided by VK_EXT_validation_features
typedef enum VkValidationFeatureDisableEXT {
    VK_VALIDATION_FEATURE_DISABLE_ALL_EXT = 0,
    VK_VALIDATION_FEATURE_DISABLE_SHADERS_EXT = 1,
    VK_VALIDATION_FEATURE_DISABLE_THREAD_SAFETY_EXT = 2,
    VK_VALIDATION_FEATURE_DISABLE_API_PARAMETERS_EXT = 3,
    VK_VALIDATION_FEATURE_DISABLE_OBJECT_LIFETIMES_EXT = 4,
    VK_VALIDATION_FEATURE_DISABLE_CORE_CHECKS_EXT = 5,
    VK_VALIDATION_FEATURE_DISABLE_UNIQUE_HANDLES_EXT = 6,
    VK_VALIDATION_FEATURE_DISABLE_SHADER_VALIDATION_CACHE_EXT = 7,
} VkValidationFeatureDisableEXT;
```

- **VK_VALIDATION_FEATURE_DISABLE_ALL_EXT** specifies that all validation checks are disabled.
- **VK_VALIDATION_FEATURE_DISABLE_SHADERS_EXT** specifies that shader validation, both runtime and standalone, is disabled. This validation occurs inside **VkShaderModuleCreateInfo**. This feature is enabled by default.
- **VK_VALIDATION_FEATURE_DISABLE_THREAD_SAFETY_EXT** specifies that thread safety validation is disabled. This feature is enabled by default.
- **VK_VALIDATION_FEATURE_DISABLE_API_PARAMETERS_EXT** specifies that stateless parameter validation is disabled. This feature is enabled by default.
- **VK_VALIDATION_FEATURE_DISABLE_OBJECT_LIFETIMES_EXT** specifies that object lifetime validation is disabled. This feature is enabled by default.
- **VK_VALIDATION_FEATURE_DISABLE_CORE_CHECKS_EXT** specifies that core validation checks are disabled. This feature is enabled by default. If this feature is disabled, **VK_VALIDATION_FEATURE_DISABLE_SHADERS_EXT** is implied.
- **VK_VALIDATION_FEATURE_DISABLE_UNIQUE_HANDLES_EXT** specifies that protection against duplicate non-dispatchable object handles is disabled. This feature is enabled by default.
- **VK_VALIDATION_FEATURE_DISABLE_SHADER_VALIDATION_CACHE_EXT** specifies that there will be no caching of shader validation results and every shader will be validated on every application execution. Shader validation caching is enabled by default.

NOTE

Disabling checks such as parameter validation and object lifetime validation prevents the reporting of error conditions that can cause other validation checks to behave incorrectly or crash. Some validation checks assume that their inputs are already valid and do not always revalidate them.

To create a Vulkan instance with a specific configuration of layer settings, add **VkLayerSettingsCreateInfoEXT** structures to the **pNext** chain of the **VkInstanceCreateInfo** structure, specifying the settings to be configured.


```
// Provided by VK_EXT_layer_settings
typedef struct VkLayerSettingsCreateInfoEXT {
    VkStructureType      sType;
    const void*          pNext;
    uint32_t             settingCount;
    const VkLayerSettingEXT* pSettings;
} VkLayerSettingsCreateInfoEXT;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **settingCount** is the number of settings to configure.
- **pSettings** is a pointer to an array of **settingCount** [VkLayerSettingEXT](#) values specifying the settings to be configured.

Valid Usage (Implicit)

- VUID-VkLayerSettingsCreateInfoEXT-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_LAYER_SETTINGS_CREATE_INFO_EXT**
- VUID-VkLayerSettingsCreateInfoEXT-pSettings-parameter
If **settingCount** is not 0, **pSettings** **must** be a valid pointer to an array of **settingCount** valid [VkLayerSettingEXT](#) structures

The values of elements of the [VkLayerSettingsCreateInfoEXT::pSettings](#) array, specifying layer settings to be configured, are:

```
// Provided by VK_EXT_layer_settings
typedef struct VkLayerSettingEXT {
    const char*      pLayerName;
    const char*      pSettingName;
    VkLayerSettingTypeEXT type;
    uint32_t         valueCount;
    const void*      pValues;
} VkLayerSettingEXT;
```

- **pLayerName** is a pointer to a null-terminated UTF-8 string naming the layer to configure the setting from.
- **pSettingName** is a pointer to a null-terminated UTF-8 string naming the setting to configure. Values of **pSettingName** that are unknown to the layer are ignored.
- **type** is a [VkLayerSettingTypeEXT](#) value specifying the type of the **pValues** values.
- **valueCount** is the number of values used to configure the layer setting.
- **pValues** is a pointer to an array of **valueCount** values of the type indicated by **type** to configure the layer setting.

When multiple `VkLayerSettingsCreateInfoEXT` structures are chained and the same `pSettingName` is referenced for the same `pLayerName`, the value of the first reference of the layer setting is used.

Valid Usage

- VUID-VkLayerSettingEXT-valueCount-10070

If `valueCount` is not 0, `pValues` **must** be a valid pointer to an array of `valueCount` values of the type indicated by `type`

Valid Usage (Implicit)

- VUID-VkLayerSettingEXT-pLayerName-parameter
`pLayerName` **must** be a null-terminated UTF-8 string
- VUID-VkLayerSettingEXT-pSettingName-parameter
`pSettingName` **must** be a null-terminated UTF-8 string
- VUID-VkLayerSettingEXT-type-parameter
`type` **must** be a valid `VkLayerSettingTypeEXT` value

Possible values of `VkLayerSettingEXT::type`, specifying the type of the data returned in `VkLayerSettingEXT::pValues`, are:

```
// Provided by VK_EXT_layer_settings
typedef enum VkLayerSettingTypeEXT {
    VK_LAYER_SETTING_TYPE_BOOL32_EXT = 0,
    VK_LAYER_SETTING_TYPE_INT32_EXT = 1,
    VK_LAYER_SETTING_TYPE_INT64_EXT = 2,
    VK_LAYER_SETTING_TYPE_UINT32_EXT = 3,
    VK_LAYER_SETTING_TYPE_UINT64_EXT = 4,
    VK_LAYER_SETTING_TYPE_FLOAT32_EXT = 5,
    VK_LAYER_SETTING_TYPE_FLOAT64_EXT = 6,
    VK_LAYER_SETTING_TYPE_STRING_EXT = 7,
} VkLayerSettingTypeEXT;
```

- `VK_LAYER_SETTING_TYPE_BOOL32_EXT` specifies that the layer setting's type is `VkBool32`.
- `VK_LAYER_SETTING_TYPE_INT32_EXT` specifies that the layer setting's type is signed 32-bit integer.
- `VK_LAYER_SETTING_TYPE_INT64_EXT` specifies that the layer setting's type is signed 64-bit integer.
- `VK_LAYER_SETTING_TYPE_UINT32_EXT` specifies that the layer setting's type is unsigned 32-bit integer.
- `VK_LAYER_SETTING_TYPE_UINT64_EXT` specifies that the layer setting's type is unsigned 64-bit integer.
- `VK_LAYER_SETTING_TYPE_FLOAT32_EXT` specifies that the layer setting's type is 32-bit floating-point.
- `VK_LAYER_SETTING_TYPE_FLOAT64_EXT` specifies that the layer setting's type is 64-bit floating-point.

- `VK_LAYER_SETTING_TYPE_STRING_EXT` specifies that the layer setting's type is a pointer to a null-terminated UTF-8 string.

The `VkApplicationInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkApplicationInfo {
    VkStructureType    sType;
    const void*        pNext;
    const char*        pApplicationName;
    uint32_t           applicationVersion;
    const char*        pEngineName;
    uint32_t           engineVersion;
    uint32_t           apiVersion;
} VkApplicationInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pApplicationName` is `NULL` or is a pointer to a null-terminated UTF-8 string containing the name of the application.
- `applicationVersion` is an unsigned integer variable containing the developer-supplied version number of the application.
- `pEngineName` is `NULL` or is a pointer to a null-terminated UTF-8 string containing the name of the engine (if any) used to create the application.
- `engineVersion` is an unsigned integer variable containing the developer-supplied version number of the engine used to create the application.
- `apiVersion` **must** be the highest version of Vulkan that the application is designed to use, encoded as described in [Version Numbers](#). The patch version number specified in `apiVersion` is ignored when creating an instance object. The variant version of the instance **must** match that requested in `apiVersion`.

Vulkan 1.0 implementations were required to return `VK_ERROR_INCOMPATIBLE_DRIVER` if `apiVersion` was larger than 1.0. Implementations that support Vulkan 1.1 or later **must** not return `VK_ERROR_INCOMPATIBLE_DRIVER` for any value of `apiVersion`, unless an incompatible variant is requested.

NOTE	Vulkan SC 1.0 is based on Vulkan 1.2 and thus instance creation may only fail with <code>VK_ERROR_INCOMPATIBLE_DRIVER</code> if an incompatible variant is requested - that is if the Vulkan SC API is requested from a Vulkan implementation or if the Vulkan API is requested from a Vulkan SC implementation.
NOTE	Providing a <code>NULL</code> <code>VkInstanceCreateInfo::pApplicationInfo</code> or providing an <code>apiVersion</code> of 0 is equivalent to providing an <code>apiVersion</code> of <code>VK_MAKE_API_VERSION(1,1,0,0)</code> .

To provide *application parameters* at instance creation time, an application **can** link one or more

`VkApplicationParametersEXT` structures to the `pNext` chain of the `VkApplicationInfo` structure.

If `VkApplicationParametersEXT::vendorID` does not correspond to an ICD that is currently available, or if `VkApplicationParametersEXT::deviceID` is not `0` and does not correspond to a physical device that is available on the system, `vkCreateInstance` will fail and return `VK_ERROR_INCOMPATIBLE_DRIVER`. If `VkApplicationParametersEXT::deviceID` is `0`, the application parameter applies to all physical devices supported by the ICD identified by `VkApplicationParametersEXT::vendorID`.

If `VkApplicationParametersEXT::key` is not a valid implementation-defined application parameter key for the instance being created with `vendorID`, or if `value` is not a valid value for the specified `key`, `vkCreateInstance` will fail and return `VK_ERROR_INITIALIZATION_FAILED`.

For any implementation-defined application parameter `key` that exists but is not set by the application, the implementation-specific default value is used.

Valid Usage

- VUID-VkApplicationInfo-apiVersion-05021
If `apiVersion` is not `0` and its variant is `VKSC_API_VARIANT`, then it **must** be greater than or equal to `VKSC_API_VERSION_1_0`
- VUID-VkApplicationInfo-key-05093
The `key` value of each `VkApplicationParametersEXT` structure in the `VkApplicationInfo::pNext` chain **must** be unique for each `vendorID` and `deviceID` pairing

Valid Usage (Implicit)

- VUID-VkApplicationInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_APPLICATION_INFO`
- VUID-VkApplicationInfo-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkApplicationParametersEXT`
- VUID-VkApplicationInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique, with the exception of structures of type `VkApplicationParametersEXT`
- VUID-VkApplicationInfo-pApplicationName-parameter
If `pApplicationName` is not `NULL`, `pApplicationName` **must** be a null-terminated UTF-8 string
- VUID-VkApplicationInfo-pEngineName-parameter
If `pEngineName` is not `NULL`, `pEngineName` **must** be a null-terminated UTF-8 string

The `VkApplicationParametersEXT` structure is defined as:

```
// Provided by VK_EXT_application_parameters
typedef struct VkApplicationParametersEXT {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           vendorID;
    uint32_t           deviceID;
    uint32_t           key;
    uint64_t           value;
} VkApplicationParametersEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `vendorID` is the `VkPhysicalDeviceProperties::vendorID` of the ICD that the application parameter is applied to.
- `deviceID` is `0` or the `VkPhysicalDeviceProperties::deviceID` of the physical device that the application parameter is applied to.
- `key` is a 32-bit vendor-specific enumerant identifying the application parameter that is being set.
- `value` is the 64-bit value that is being set for the application parameter specified by `key`.

Valid Usage (Implicit)

- VUID-VkApplicationParametersEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_APPLICATION_PARAMETERS_EXT`

To destroy an instance, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyInstance(
    VkInstance          instance,
    const VkAllocationCallbacks* pAllocator);
```

- `instance` is the handle of the instance to destroy.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.

Prior to destroying an instance, an application is responsible for destroying/freeing any Vulkan objects with explicit `vkDestroy*` or `vkFree*` commands that were created using that instance, or any `VkPhysicalDevice` object retrieved from it, as the first parameter of the corresponding `vkCreate*` or `vkAllocate*` command.

Valid Usage

- VUID-vkDestroyInstance-instance-00629
All child objects that were created with `instance` or with a `VkPhysicalDevice` retrieved

from it, and that **can** be destroyed or freed, **must** have been destroyed or freed prior to destroying **instance**

Valid Usage (Implicit)

- VUID-vkDestroyInstance-instance-parameter
If **instance** is not **NULL**, **instance** **must** be a valid **VkInstance** handle
- VUID-vkDestroyInstance-pAllocator-null
pAllocator **must** be **NULL**

Host Synchronization

- Host access to **instance** **must** be externally synchronized
- Host access to all **VkPhysicalDevice** objects enumerated from **instance** **must** be externally synchronized

Chapter 5. Devices and Queues

Once Vulkan is initialized, devices and queues are the primary objects used to interact with a Vulkan implementation.

Vulkan separates the concept of *physical* and *logical* devices. A physical device usually represents a single complete implementation of Vulkan (excluding instance-level functionality) available to the host, of which there are a finite number. A logical device represents an instance of that implementation with its own state and resources independent of other logical devices.

Physical devices **cannot** be independently destroyed, and are instead destroyed with the [VkInstance](#) that they were retrieved from.

Physical devices are represented by [VkPhysicalDevice](#) handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_HANDLE(VkPhysicalDevice)
```

5.1. Physical Devices

To retrieve a list of physical device objects representing the physical devices installed in the system, call:

```
// Provided by VK_VERSION_1_0
VkResult vkEnumeratePhysicalDevices(
    VkInstance          instance,
    uint32_t*           pPhysicalDeviceCount,
    VkPhysicalDevice*   pPhysicalDevices);
```

- [instance](#) is a handle to a Vulkan instance previously created with [vkCreateInstance](#).
- [pPhysicalDeviceCount](#) is a pointer to an integer related to the number of physical devices available or queried, as described below.
- [pPhysicalDevices](#) is either [NULL](#) or a pointer to an array of [VkPhysicalDevice](#) handles.

If [pPhysicalDevices](#) is [NULL](#), then the number of physical devices available is returned in [pPhysicalDeviceCount](#). Otherwise, [pPhysicalDeviceCount](#) **must** point to a variable set by the application to the number of elements in the [pPhysicalDevices](#) array, and on return the variable is overwritten with the number of handles actually written to [pPhysicalDevices](#). If [pPhysicalDeviceCount](#) is less than the number of physical devices available, at most [pPhysicalDeviceCount](#) structures will be written, and [VK_INCOMPLETE](#) will be returned instead of [VK_SUCCESS](#), to indicate that not all the available physical devices were returned.

Valid Usage (Implicit)

- VUID-vkEnumeratePhysicalDevices-instance-parameter

`instance` **must** be a valid [VkInstance](#) handle

- VUID-vkEnumeratePhysicalDevices-pPhysicalDeviceCount-parameter

`pPhysicalDeviceCount` **must** be a valid pointer to a `uint32_t` value

- VUID-vkEnumeratePhysicalDevices-pPhysicalDevices-parameter

If the value referenced by `pPhysicalDeviceCount` is not `0`, and `pPhysicalDevices` is not `NULL`, `pPhysicalDevices` **must** be a valid pointer to an array of `pPhysicalDeviceCount` [VkPhysicalDevice](#) handles

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To query general properties of physical devices once enumerated, call:

WARNING

This functionality is superseded by [Vulkan Version 1.1](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
void vkGetPhysicalDeviceProperties(
    VkPhysicalDevice          physicalDevice,
    VkPhysicalDeviceProperties* pProperties);
```

- `physicalDevice` is the handle to the physical device whose properties will be queried.
- `pProperties` is a pointer to a [VkPhysicalDeviceProperties](#) structure in which properties are returned.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceProperties-physicalDevice-parameter

`physicalDevice` **must** be a valid [VkPhysicalDevice](#) handle

- VUID-vkGetPhysicalDeviceProperties-pProperties-parameter

`pProperties` **must** be a valid pointer to a [VkPhysicalDeviceProperties](#) structure

The `VkPhysicalDeviceProperties` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPhysicalDeviceProperties {
    uint32_t          apiVersion;
    uint32_t          driverVersion;
    uint32_t          vendorID;
    uint32_t          deviceID;
    VkPhysicalDeviceType deviceType;
    char              deviceName[VK_MAX_PHYSICAL_DEVICE_NAME_SIZE];
    uint8_t           pipelineCacheUUID[VK_UUID_SIZE];
    VkPhysicalDeviceLimits limits;
    VkPhysicalDeviceSparseProperties sparseProperties;
} VkPhysicalDeviceProperties;
```

- `apiVersion` is the version of Vulkan supported by the device, encoded as described in [Version Numbers](#).
- `driverVersion` is the vendor-specified version of the driver.
- `vendorID` is a unique identifier for the *vendor* (see below) of the physical device.
- `deviceID` is a unique identifier for the physical device among devices available from the vendor.
- `deviceType` is a [VkPhysicalDeviceType](#) specifying the type of device.
- `deviceName` is an array of `VK_MAX_PHYSICAL_DEVICE_NAME_SIZE` `char` containing a null-terminated UTF-8 string which is the name of the device.
- `pipelineCacheUUID` is an array of `VK_UUID_SIZE` `uint8_t` values representing a universally unique identifier for the device.
- `limits` is the [VkPhysicalDeviceLimits](#) structure specifying device-specific limits of the physical device. See [Limits](#) for details.
- `sparseProperties` is the [VkPhysicalDeviceSparseProperties](#) structure specifying various sparse related properties of the physical device. See [Sparse Properties](#) for details.

NOTE The value of `apiVersion` **may** be different than the version returned by [vkEnumerateInstanceVersion](#); either higher or lower. In such cases, the application **must** not use functionality that exceeds the version of Vulkan associated with a given object. The `pApiVersion` parameter returned by [vkEnumerateInstanceVersion](#) is the version associated with a [VkInstance](#) and its children, except for a [VkPhysicalDevice](#) and its children. `VkPhysicalDeviceProperties::apiVersion` is the version associated with a [VkPhysicalDevice](#) and its children.

NOTE The encoding of `driverVersion` is implementation-defined. It **may** not use the same encoding as `apiVersion`. Applications should follow information from the *vendor* on how to extract the version information from `driverVersion`.

The `vendorID` and `deviceID` fields are provided to allow applications to adapt to device characteristics that are not adequately exposed by other Vulkan queries.

NOTE These **may** include performance profiles, hardware errata, or other characteristics.

The *vendor* identified by **vendorID** is the entity responsible for the most salient characteristics of the underlying implementation of the **VkPhysicalDevice** being queried.

NOTE For example, in the case of a discrete GPU implementation, this **should** be the GPU chipset vendor. In the case of a hardware accelerator integrated into a system-on-chip (SoC), this **should** be the supplier of the silicon IP used to create the accelerator.

If the vendor has a **PCI vendor ID**, the low 16 bits of **vendorID** **must** contain that PCI vendor ID, and the remaining bits **must** be zero. Otherwise, the value returned **must** be a valid Khronos vendor ID, obtained as described in the [Vulkan Documentation and Extensions: Procedures and Conventions](#) document in the section “Registering a Vendor ID with Khronos”. Khronos vendor IDs are allocated starting at 0x10000, to distinguish them from the PCI vendor ID namespace. Khronos vendor IDs are symbolically defined in the **VkVendorId** type.

The vendor is also responsible for the value returned in **deviceID**. If the implementation is driven primarily by a **PCI device** with a **PCI device ID**, the low 16 bits of **deviceID** **must** contain that PCI device ID, and the remaining bits **must** be zero. Otherwise, the choice of what values to return **may** be dictated by operating system or platform policies - but **should** uniquely identify both the device version and any major configuration options (for example, core count in the case of multicore devices).

NOTE The same device ID **should** be used for all physical implementations of that device version and configuration. For example, all uses of a specific silicon IP GPU version and configuration **should** use the same device ID, even if those uses occur in different SoCs.

Khronos vendor IDs which **may** be returned in **VkPhysicalDeviceProperties::vendorID** are:

```
// Provided by VK_VERSION_1_0
typedef enum VkVendorId {
    VK_VENDOR_ID_KHRONOS = 0x10000,
    VK_VENDOR_ID_VIV = 0x10001,
    VK_VENDOR_ID_VSI = 0x10002,
    VK_VENDOR_ID_KAZAN = 0x10003,
    VK_VENDOR_ID_CODEPLAY = 0x10004,
    VK_VENDOR_ID_MESA = 0x10005,
    VK_VENDOR_ID_POCL = 0x10006,
    VK_VENDOR_ID_MOBILEYE = 0x10007,
} VkVendorId;
```

NOTE Khronos vendor IDs may be allocated by vendors at any time. Only the latest canonical versions of this Specification, of the corresponding **vk.xml** API Registry, and of the corresponding **vulkan_sc_core.h** header file **must** contain all reserved Khronos vendor IDs.

Only Khronos vendor IDs are given symbolic names at present. PCI vendor IDs returned by the implementation can be looked up in the PCI-SIG database.

`VK_MAX_PHYSICAL_DEVICE_NAME_SIZE` is the length in `char` values of an array containing a physical device name string, as returned in `VkPhysicalDeviceProperties::deviceName`.

```
#define VK_MAX_PHYSICAL_DEVICE_NAME_SIZE 256U
```

The physical device types which **may** be returned in `VkPhysicalDeviceProperties::deviceType` are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPhysicalDeviceType {
    VK_PHYSICAL_DEVICE_TYPE_OTHER = 0,
    VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU = 1,
    VK_PHYSICAL_DEVICE_TYPE_DISCRETE_GPU = 2,
    VK_PHYSICAL_DEVICE_TYPE_VIRTUAL_GPU = 3,
    VK_PHYSICAL_DEVICE_TYPE_CPU = 4,
} VkPhysicalDeviceType;
```

- `VK_PHYSICAL_DEVICE_TYPE_OTHER` - the device does not match any other available types.
- `VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU` - the device is typically one embedded in or tightly coupled with the host.
- `VK_PHYSICAL_DEVICE_TYPE_DISCRETE_GPU` - the device is typically a separate processor connected to the host via an interlink.
- `VK_PHYSICAL_DEVICE_TYPE_VIRTUAL_GPU` - the device is typically a virtual node in a virtualization environment.
- `VK_PHYSICAL_DEVICE_TYPE_CPU` - the device is typically running on the same processors as the host.

The physical device type is advertised for informational purposes only, and does not directly affect the operation of the system. However, the device type **may** correlate with other advertised properties or capabilities of the system, such as how many memory heaps there are.

To query general properties of physical devices once enumerated, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceProperties2(
    VkPhysicalDevice          physicalDevice,
    VkPhysicalDeviceProperties2* pProperties);
```

- `physicalDevice` is the handle to the physical device whose properties will be queried.
- `pProperties` is a pointer to a `VkPhysicalDeviceProperties2` structure in which properties are returned.

Each structure in `pProperties` and its `pNext` chain contains members corresponding to

implementation-dependent properties, behaviors, or limits. `vkGetPhysicalDeviceProperties2` fills in each member to specify the corresponding value for the implementation.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceProperties2-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceProperties2-pProperties-parameter `pProperties` **must** be a valid pointer to a `VkPhysicalDeviceProperties2` structure

The `VkPhysicalDeviceProperties2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceProperties2 {
    VkStructureType    sType;
    void*              pNext;
    VkPhysicalDeviceProperties  properties;
} VkPhysicalDeviceProperties2;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `properties` is a `VkPhysicalDeviceProperties` structure describing properties of the physical device. This structure is written with the same values as if it were written by `vkGetPhysicalDeviceProperties`.

The `pNext` chain of this structure is used to extend the structure with properties defined by extensions.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceProperties2-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROPERTIES_2`
- VUID-VkPhysicalDeviceProperties2-pNext-pNext
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT`, `VkPhysicalDeviceConservativeRasterizationPropertiesEXT`, `VkPhysicalDeviceCustomBorderColorPropertiesEXT`, `VkPhysicalDeviceDepthStencilResolveProperties`, `VkPhysicalDeviceDescriptorIndexingProperties`, `VkPhysicalDeviceDiscardRectanglePropertiesEXT`, `VkPhysicalDeviceDriverProperties`, `VkPhysicalDeviceExternalMemoryHostPropertiesEXT`, `VkPhysicalDeviceFloatControlsProperties`, `VkPhysicalDeviceFragmentShadingRatePropertiesKHR`, `VkPhysicalDeviceIDProperties`, `VkPhysicalDeviceLineRasterizationProperties`, `VkPhysicalDeviceMaintenance3Properties`,

[VkPhysicalDeviceMultiviewProperties](#), [VkPhysicalDevicePCIBusInfoPropertiesEXT](#),
[VkPhysicalDevicePerformanceQueryPropertiesKHR](#),
[VkPhysicalDevicePointClippingProperties](#), [VkPhysicalDeviceProtectedMemoryProperties](#),
[VkPhysicalDeviceRobustness2PropertiesKHR](#),
[VkPhysicalDeviceSampleLocationsPropertiesEXT](#),
[VkPhysicalDeviceSamplerFilterMinmaxProperties](#), [VkPhysicalDeviceSubgroupProperties](#),
[VkPhysicalDeviceSubgroupSizeControlProperties](#),
[VkPhysicalDeviceTexelBufferAlignmentProperties](#),
[VkPhysicalDeviceTimelineSemaphoreProperties](#),
[VkPhysicalDeviceVertexAttributeDivisorProperties](#), [VkPhysicalDeviceVulkan11Properties](#),
[VkPhysicalDeviceVulkan12Properties](#), or [VkPhysicalDeviceVulkanSC10Properties](#)

- `VUID-VkPhysicalDeviceProperties2-sType-unique`
The `sType` value of each structure in the `pNext` chain **must** be unique

The `VkPhysicalDeviceVulkan11Properties` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceVulkan11Properties {
    VkStructureType    sType;
    void*              pNext;
    uint8_t             deviceUUID[VK_UUID_SIZE];
    uint8_t             driverUUID[VK_UUID_SIZE];
    uint8_t             deviceLUID[VK_LUID_SIZE];
    uint32_t            deviceNodeMask;
    VkBool32            deviceLUIDValid;
    uint32_t            subgroupSize;
    VkShaderStageFlags  subgroupSupportedStages;
    VkSubgroupFeatureFlags subgroupSupportedOperations;
    VkBool32            subgroupQuadOperationsInAllStages;
    VkPointClippingBehavior pointClippingBehavior;
    uint32_t            maxMultiviewViewCount;
    uint32_t            maxMultiviewInstanceIndex;
    VkBool32            protectedNoFault;
    uint32_t            maxPerSetDescriptors;
    VkDeviceSize         maxMemoryAllocationSize;
} VkPhysicalDeviceVulkan11Properties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `deviceUUID` is an array of `VK_UUID_SIZE` `uint8_t` values representing a universally unique identifier for the device.
- `driverUUID` is an array of `VK_UUID_SIZE` `uint8_t` values representing a universally unique identifier for the driver build in use by the device.
- `deviceLUID` is an array of `VK_LUID_SIZE` `uint8_t` values representing a locally unique identifier for the device.

- `deviceNodeMask` is a `uint32_t` bitfield identifying the node within a linked device adapter corresponding to the device.
- `deviceLUIDValid` is a boolean value that will be `VK_TRUE` if `deviceLUID` contains a valid LUID and `deviceNodeMask` contains a valid node mask, and `VK_FALSE` if they do not.
- `subgroupSize` is the default number of invocations in each subgroup. `subgroupSize` is at least 1 if any of the physical device's queues support `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`. `subgroupSize` is a power-of-two.
- `subgroupSupportedStages` is a bitfield of `VkShaderStageFlagBits` describing the shader stages that `group operations` with `subgroup scope` are supported in. `subgroupSupportedStages` will have the `VK_SHADER_STAGE_COMPUTE_BIT` bit set if any of the physical device's queues support `VK_QUEUE_COMPUTE_BIT`.
- `subgroupSupportedOperations` is a bitmask of `VkSubgroupFeatureFlagBits` specifying the sets of `group operations` with `subgroup scope` supported on this device. `subgroupSupportedOperations` will have the `VK_SUBGROUP_FEATURE_BASIC_BIT` bit set if any of the physical device's queues support `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`.
- `subgroupQuadOperationsInAllStages` is a boolean specifying whether `quad group operations` are available in all stages, or are restricted to fragment and compute stages.
- `pointClippingBehavior` is a `VkPointClippingBehavior` value specifying the point clipping behavior supported by the implementation.
- `maxMultiviewViewCount` is one greater than the maximum view index that **can** be used in a subpass.
- `maxMultiviewInstanceIndex` is the maximum valid value of instance index allowed to be generated by a drawing command recorded within a subpass of a multiview render pass instance.
- `protectedNoFault` specifies how an implementation behaves when an application attempts to write to unprotected memory in a protected queue operation, read from protected memory in an unprotected queue operation, or perform a query in a protected queue operation. If this limit is `VK_TRUE`, such writes will be discarded or have undefined values written, reads and queries will return undefined values. If this limit is `VK_FALSE`, applications **must** not perform these operations. See [Protected Memory Access Rules](#) for more information.
- `maxPerSetDescriptors` is a maximum number of descriptors (summed over all descriptor types) in a single descriptor set that is guaranteed to satisfy any implementation-dependent constraints on the size of a descriptor set itself. Applications **can** query whether a descriptor set that goes beyond this limit is supported using `vkGetDescriptorSetLayoutSupport`.
- `maxMemoryAllocationSize` is the maximum size of a memory allocation that **can** be created, even if there is more space available in the heap. If `VkMemoryAllocateInfo::allocationSize` is larger the error `VK_ERROR_OUT_OF_DEVICE_MEMORY` **may** be returned.

If the `VkPhysicalDeviceVulkan11Properties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

These properties correspond to Vulkan 1.1 functionality.

The members of `VkPhysicalDeviceVulkan11Properties` have the same values as the corresponding members of `VkPhysicalDeviceIDProperties`, `VkPhysicalDeviceSubgroupProperties`, `VkPhysicalDevicePointClippingProperties`, `VkPhysicalDeviceMultiviewProperties`, `VkPhysicalDeviceProtectedMemoryProperties`, and `VkPhysicalDeviceMaintenance3Properties`.

NOTE

The `subgroupSupportedStages`, `subgroupSupportedOperations`, and `subgroupQuadOperationsInAllStages` members of this structure correspond respectively to the `VkPhysicalDeviceSubgroupProperties::supportedStages`, `VkPhysicalDeviceSubgroupProperties::supportedOperations`, and `VkPhysicalDeviceSubgroupProperties::quadOperationsInAllStages` members, but add the `subgroup` prefix to the member name.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVulkan11Properties-sType-sType
`sType` must be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_1_PROPERTIES`

The `VkPhysicalDeviceVulkan12Properties` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceVulkan12Properties {
    VkStructureType    sType;
    void*              pNext;
    VkDriverId         driverID;
    char               driverName[VK_MAX_DRIVER_NAME_SIZE];
    char               driverInfo[VK_MAX_DRIVER_INFO_SIZE];
    VkConformanceVersion conformanceVersion;
    VkShaderFloatControlsIndependence denormBehaviorIndependence;
    VkShaderFloatControlsIndependence roundingModeIndependence;
    VkBool32           shaderSignedZeroInfNanPreserveFloat16;
    VkBool32           shaderSignedZeroInfNanPreserveFloat32;
    VkBool32           shaderSignedZeroInfNanPreserveFloat64;
    VkBool32           shaderDenormPreserveFloat16;
    VkBool32           shaderDenormPreserveFloat32;
    VkBool32           shaderDenormPreserveFloat64;
    VkBool32           shaderDenormFlushToZeroFloat16;
    VkBool32           shaderDenormFlushToZeroFloat32;
    VkBool32           shaderDenormFlushToZeroFloat64;
    VkBool32           shaderRoundingModeRTEFloat16;
    VkBool32           shaderRoundingModeRTEFloat32;
    VkBool32           shaderRoundingModeRTEFloat64;
    VkBool32           shaderRoundingModeRTZFloat16;
    VkBool32           shaderRoundingModeRTZFloat32;
    VkBool32           shaderRoundingModeRTZFloat64;
    uint32_t           maxUpdateAfterBindDescriptorsInAllPools;
    VkBool32
    shaderUniformBufferArrayNonUniformIndexingNative;
    VkBool32
```

```

shaderSampledImageArrayNonUniformIndexingNative;
    VkBool32
shaderStorageBufferArrayNonUniformIndexingNative;
    VkBool32
shaderStorageImageArrayNonUniformIndexingNative;
    VkBool32
shaderInputAttachmentArrayNonUniformIndexingNative;
    VkBool32
    robustBufferAccessUpdateAfterBind;
    VkBool32
    quadDivergentImplicitLod;
    uint32_t
    uint32_t
    maxPerStageDescriptorUpdateAfterBindUniformBuffers;
    uint32_t
maxPerStageDescriptorUpdateAfterBindStorageBuffers;
    uint32_t
maxPerStageDescriptorUpdateAfterBindSampledImages;
    uint32_t
maxPerStageDescriptorUpdateAfterBindStorageImages;
    uint32_t
maxPerStageDescriptorUpdateAfterBindInputAttachments;
    uint32_t
    maxPerStageUpdateAfterBindResources;
    uint32_t
    maxDescriptorSetUpdateAfterBindSamplers;
    uint32_t
maxDescriptorSetUpdateAfterBindUniformBuffers;
    uint32_t
maxDescriptorSetUpdateAfterBindUniformBuffersDynamic;
    uint32_t
maxDescriptorSetUpdateAfterBindStorageBuffers;
    uint32_t
maxDescriptorSetUpdateAfterBindStorageBuffersDynamic;
    uint32_t
    maxDescriptorSetUpdateAfterBindSampledImages;
    uint32_t
    maxDescriptorSetUpdateAfterBindStorageImages;
    uint32_t
maxDescriptorSetUpdateAfterBindInputAttachments;
    VkResolveModeFlags
    supportedDepthResolveModes;
    VkResolveModeFlags
    supportedStencilResolveModes;
    VkBool32
    independentResolveNone;
    VkBool32
    independentResolve;
    VkBool32
    filterMinmaxSingleComponentFormats;
    VkBool32
    filterMinmaxImageComponentMapping;
    uint64_t
    maxTimelineSemaphoreValueDifference;
    VkSampleCountFlags
    framebufferIntegerColorSampleCounts;
} VkPhysicalDeviceVulkan12Properties;

```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is `NULL` or a pointer to a structure extending this structure.
- **driverID** is a unique identifier for the driver of the physical device.
- **driverName** is an array of `VK_MAX_DRIVER_NAME_SIZE` `char` containing a null-terminated UTF-8 string

which is the name of the driver.

- `driverInfo` is an array of `VK_MAX_DRIVER_INFO_SIZE` `char` containing a null-terminated UTF-8 string with additional information about the driver.
- `conformanceVersion` is the latest version of the Vulkan conformance test that the implementor has successfully tested this driver against prior to release (see [VkConformanceVersion](#)).
- `denormBehaviorIndependence` is a [VkShaderFloatControlsIndependence](#) value indicating whether, and how, denorm behavior can be set independently for different bit widths.
- `roundingModeIndependence` is a [VkShaderFloatControlsIndependence](#) value indicating whether, and how, rounding modes can be set independently for different bit widths.
- `shaderSignedZeroInfNanPreserveFloat16` is a boolean value indicating whether sign of a zero, Nans and $\pm\infty$ **can** be preserved in 16-bit floating-point computations. It also indicates whether the `SignedZeroInfNanPreserve` execution mode **can** be used for 16-bit floating-point types.
- `shaderSignedZeroInfNanPreserveFloat32` is a boolean value indicating whether sign of a zero, Nans and $\pm\infty$ **can** be preserved in 32-bit floating-point computations. It also indicates whether the `SignedZeroInfNanPreserve` execution mode **can** be used for 32-bit floating-point types.
- `shaderSignedZeroInfNanPreserveFloat64` is a boolean value indicating whether sign of a zero, Nans and $\pm\infty$ **can** be preserved in 64-bit floating-point computations. It also indicates whether the `SignedZeroInfNanPreserve` execution mode **can** be used for 64-bit floating-point types.
- `shaderDenormPreserveFloat16` is a boolean value indicating whether denormals **can** be preserved in 16-bit floating-point computations. It also indicates whether the `DenormPreserve` execution mode **can** be used for 16-bit floating-point types.
- `shaderDenormPreserveFloat32` is a boolean value indicating whether denormals **can** be preserved in 32-bit floating-point computations. It also indicates whether the `DenormPreserve` execution mode **can** be used for 32-bit floating-point types.
- `shaderDenormPreserveFloat64` is a boolean value indicating whether denormals **can** be preserved in 64-bit floating-point computations. It also indicates whether the `DenormPreserve` execution mode **can** be used for 64-bit floating-point types.
- `shaderDenormFlushToZeroFloat16` is a boolean value indicating whether denormals **can** be flushed to zero in 16-bit floating-point computations. It also indicates whether the `DenormFlushToZero` execution mode **can** be used for 16-bit floating-point types.
- `shaderDenormFlushToZeroFloat32` is a boolean value indicating whether denormals **can** be flushed to zero in 32-bit floating-point computations. It also indicates whether the `DenormFlushToZero` execution mode **can** be used for 32-bit floating-point types.
- `shaderDenormFlushToZeroFloat64` is a boolean value indicating whether denormals **can** be flushed to zero in 64-bit floating-point computations. It also indicates whether the `DenormFlushToZero` execution mode **can** be used for 64-bit floating-point types.
- `shaderRoundingModeRTEFloat16` is a boolean value indicating whether an implementation supports the round-to-nearest-even rounding mode for 16-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTE` execution mode **can** be used for 16-bit floating-point types.
- `shaderRoundingModeRTEFloat32` is a boolean value indicating whether an implementation supports the round-to-nearest-even rounding mode for 32-bit floating-point arithmetic and

conversion instructions. It also indicates whether the `RoundingModeRTE` execution mode **can** be used for 32-bit floating-point types.

- `shaderRoundingModeRTEFloat64` is a boolean value indicating whether an implementation supports the round-to-nearest-even rounding mode for 64-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTE` execution mode **can** be used for 64-bit floating-point types.
- `shaderRoundingModeRTZFloat16` is a boolean value indicating whether an implementation supports the round-towards-zero rounding mode for 16-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTZ` execution mode **can** be used for 16-bit floating-point types.
- `shaderRoundingModeRTZFloat32` is a boolean value indicating whether an implementation supports the round-towards-zero rounding mode for 32-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTZ` execution mode **can** be used for 32-bit floating-point types.
- `shaderRoundingModeRTZFloat64` is a boolean value indicating whether an implementation supports the round-towards-zero rounding mode for 64-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTZ` execution mode **can** be used for 64-bit floating-point types.
- `maxUpdateAfterBindDescriptorsInAllPools` is the maximum number of descriptors (summed over all descriptor types) that **can** be created across all pools that are created with the `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT` bit set. Pool creation **may** fail when this limit is exceeded, or when the space this limit represents is unable to satisfy a pool creation due to fragmentation.
- `shaderUniformBufferArrayNonUniformIndexingNative` is a boolean value indicating whether uniform buffer descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of uniform buffers **may** execute multiple times in order to access all the descriptors.
- `shaderSampledImageArrayNonUniformIndexingNative` is a boolean value indicating whether sampler and image descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of samplers or images **may** execute multiple times in order to access all the descriptors.
- `shaderStorageBufferArrayNonUniformIndexingNative` is a boolean value indicating whether storage buffer descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of storage buffers **may** execute multiple times in order to access all the descriptors.
- `shaderStorageImageArrayNonUniformIndexingNative` is a boolean value indicating whether storage image descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of storage images **may** execute multiple times in order to access all the descriptors.
- `shaderInputAttachmentArrayNonUniformIndexingNative` is a boolean value indicating whether input attachment descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of input attachments **may** execute multiple times in order to access all the descriptors.

- `robustBufferAccessUpdateAfterBind` is a boolean value indicating whether `robustBufferAccess` **can** be enabled on a device simultaneously with `descriptorBindingUniformBufferUpdateAfterBind`, `descriptorBindingStorageBufferUpdateAfterBind`, `descriptorBindingUniformTexelBufferUpdateAfterBind`, and/or `descriptorBindingStorageTexelBufferUpdateAfterBind`. If this is `VK_FALSE`, then either `robustBufferAccess` **must** be disabled or all of these update-after-bind features **must** be disabled.
- `quadDivergentImplicitLod` is a boolean value indicating whether implicit LOD calculations for image operations have well-defined results when the image and/or sampler objects used for the instruction are not uniform within a quad. See [Derivative Image Operations](#).
- `maxPerStageDescriptorUpdateAfterBindSamplers` is similar to `maxPerStageDescriptorSamplers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindUniformBuffers` is similar to `maxPerStageDescriptorUniformBuffers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindStorageBuffers` is similar to `maxPerStageDescriptorStorageBuffers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindSampledImages` is similar to `maxPerStageDescriptorSampledImages` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindStorageImages` is similar to `maxPerStageDescriptorStorageImages` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindInputAttachments` is similar to `maxPerStageDescriptorInputAttachments` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageUpdateAfterBindResources` is similar to `maxPerStageResources` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxDescriptorSetUpdateAfterBindSamplers` is similar to `maxDescriptorSetSamplers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxDescriptorSetUpdateAfterBindUniformBuffers` is similar to `maxDescriptorSetUniformBuffers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxDescriptorSetUpdateAfterBindUniformBuffersDynamic` is similar to `maxDescriptorSetUniformBuffersDynamic` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set. While an application **can** allocate dynamic uniform buffer descriptors from a pool created with the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT`, bindings for these descriptors **must** not be present in any descriptor set layout that includes bindings created with `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`.

- `maxDescriptorSetUpdateAfterBindStorageBuffers` is similar to `maxDescriptorSetStorageBuffers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxDescriptorSetUpdateAfterBindStorageBuffersDynamic` is similar to `maxDescriptorSetStorageBuffersDynamic` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set. While an application **can** allocate dynamic storage buffer descriptors from a pool created with the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT`, bindings for these descriptors **must** not be present in any descriptor set layout that includes bindings created with `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`.
- `maxDescriptorSetUpdateAfterBindSampledImages` is similar to `maxDescriptorSetSampledImages` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxDescriptorSetUpdateAfterBindStorageImages` is similar to `maxDescriptorSetStorageImages` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxDescriptorSetUpdateAfterBindInputAttachments` is similar to `maxDescriptorSetInputAttachments` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `supportedDepthResolveModes` is a bitmask of `VkResolveModeFlagBits` indicating the set of supported depth resolve modes. A value of `VK_RESOLVE_MODE_NONE` indicates that depth resolve operations are disallowed [SCID-8]. If any bits are set then `VK_RESOLVE_MODE_SAMPLE_ZERO_BIT` **must** be included in the set but implementations **may** support additional modes.
- `supportedStencilResolveModes` is a bitmask of `VkResolveModeFlagBits` indicating the set of supported stencil resolve modes. A value of `VK_RESOLVE_MODE_NONE` indicates that stencil resolve operations are disallowed [SCID-8]. If any bits are set then `VK_RESOLVE_MODE_SAMPLE_ZERO_BIT` **must** be included in the set but implementations **may** support additional modes. `VK_RESOLVE_MODE_AVERAGE_BIT` **must** not be included in the set.
- `independentResolveNone` is `VK_TRUE` if the implementation supports setting the depth and stencil resolve modes to different values when one of those modes is `VK_RESOLVE_MODE_NONE`. Otherwise the implementation only supports setting both modes to the same value.
- `independentResolve` is `VK_TRUE` if the implementation supports all combinations of the supported depth and stencil resolve modes, including setting either depth or stencil resolve mode to `VK_RESOLVE_MODE_NONE`. An implementation that supports `independentResolve` **must** also support `independentResolveNone`.
- `filterMinmaxSingleComponentFormats` is a boolean value indicating whether a minimum set of required formats support min/max filtering.
- `filterMinmaxImageComponentMapping` is a boolean value indicating whether the implementation supports non-identity component mapping of the image when doing min/max filtering.
- `maxTimelineSemaphoreValueDifference` indicates the maximum difference allowed by the implementation between the current value of a timeline semaphore and any pending signal or wait operations.
- `framebufferIntegerColorSampleCounts` is a bitmask of `VkSampleCountFlagBits` indicating the color

sample counts that are supported for all framebuffer color attachments with integer formats.

If the `VkPhysicalDeviceVulkan12Properties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

These properties correspond to Vulkan 1.2 functionality.

The members of `VkPhysicalDeviceVulkan12Properties` **must** have the same values as the corresponding members of `VkPhysicalDeviceDriverProperties`, `VkPhysicalDeviceFloatControlsProperties`, `VkPhysicalDeviceDescriptorIndexingProperties`, `VkPhysicalDeviceDepthStencilResolveProperties`, `VkPhysicalDeviceSamplerFilterMinmaxProperties`, and `VkPhysicalDeviceTimelineSemaphoreProperties`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVulkan12Properties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_2_PROPERTIES`

The `VkPhysicalDeviceVulkanSC10Properties` structure is defined as:

```
// Provided by VKSC_VERSION_1_0
typedef struct VkPhysicalDeviceVulkanSC10Properties {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           deviceNoDynamicHostAllocations;
    VkBool32           deviceDestroyFreesMemory;
    VkBool32           commandPoolMultipleCommandBuffersRecording;
    VkBool32           commandPoolResetCommandBuffer;
    VkBool32           commandBufferSimultaneousUse;
    VkBool32           secondaryCommandBufferNullOrImagelessFramebuffer;
    VkBool32           recycleDescriptorSetMemory;
    VkBool32           recyclePipelineMemory;
    uint32_t           maxRenderPassSubpasses;
    uint32_t           maxRenderPassDependencies;
    uint32_t           maxSubpassInputAttachments;
    uint32_t           maxSubpassPreserveAttachments;
    uint32_t           maxFramebufferAttachments;
    uint32_t           maxDescriptorSetLayoutBindings;
    uint32_t           maxQueryFaultCount;
    uint32_t           maxCallbackFaultCount;
    uint32_t           maxCommandPoolCommandBuffers;
    VkDeviceSize       maxCommandBufferSize;
} VkPhysicalDeviceVulkanSC10Properties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- `deviceNoDynamicHostAllocations` indicates whether the implementation will perform dynamic host memory allocations for physical or logical device commands. If `deviceNoDynamicHostAllocations` is `VK_TRUE` the implementation will allocate host memory for objects based on the provided `VkDeviceObjectReservationCreateInfo` limits during `vkCreateDevice`. Under valid API usage, `VK_ERROR_OUT_OF_HOST_MEMORY` **may** only be returned by commands which do not explicitly disallow it.
- `deviceDestroyFreesMemory` indicates whether destroying the device frees all memory resources back to the system.
- `commandPoolMultipleCommandBuffersRecording` indicates whether multiple command buffers from the same command pool **can** be in the `recording state` at the same time.
- `commandPoolResetCommandBuffer` indicates whether command buffers support `vkResetCommandBuffer`, and `vkBeginCommandBuffer` when not in the `initial state`.
- `commandBufferSimultaneousUse` indicates whether command buffers support `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT`.
- `secondaryCommandBufferNullOrImagelessFramebuffer` indicates whether the `framebuffer` member of `VkCommandBufferInheritanceInfo` **may** be equal to `VK_NULL_HANDLE` or be created with a `VkFramebufferCreateInfo::flags` value that includes `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT` if the command buffer will be executed within a render pass instance.
- `recycleDescriptorSetMemory` indicates whether descriptor pools are able to immediately reuse pool memory from descriptor sets that have been freed. If this is `VK_FALSE`, then memory **may** only be reallocated after `vkResetDescriptorPool` is called.
- `recyclePipelineMemory` indicates whether the memory for a pipeline is available for reuse by new pipelines after the pipeline is destroyed.
- `maxRenderPassSubpasses` is the maximum number of subpasses in a render pass.
- `maxRenderPassDependencies` is the maximum number of dependencies in a render pass.
- `maxSubpassInputAttachments` is the maximum number of input attachments in a subpass.
- `maxSubpassPreserveAttachments` is the maximum number of preserve attachments in a subpass.
- `maxFramebufferAttachments` is the maximum number of attachments in a framebuffer, as well as the maximum number of attachments in a render pass.
- `maxDescriptorSetLayoutBindings` is the maximum number of bindings in a descriptor set layout.
- `maxQueryFaultCount` is the maximum number of faults that the implementation **can** record, to be reported via `vkGetFaultData`.
- `maxCallbackFaultCount` is the maximum number of faults that the implementation **can** report via a single call to `PFN_vkFaultCallbackFunction`.
- `maxCommandPoolCommandBuffers` is the maximum number of command buffers that **can** be allocated from a single command pool.
- `maxCommandBufferSize` is the maximum supported size of a single command buffer in bytes. Applications **can** use `vkGetCommandPoolMemoryConsumption` to compare a command buffer's current memory usage to this limit.

NOTE | Implementations that do not have a fixed upper bound on the number of command

buffers that **may** be allocated from a command pool **can** report 0xFFFFFFFFU for `maxCommandPoolCommandBuffers`.

Implementations that do not have a fixed upper bound on the command buffer size **can** report `UINT64_MAX` for `maxCommandBufferSize`.

If the `VkPhysicalDeviceVulkanSC10Properties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

These properties correspond to Vulkan SC 1.0 functionality.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceVulkanSC10Properties-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_SC_1_0_PROPERTIES`

The `VkPhysicalDeviceIDProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceIDProperties {
    VkStructureType    sType;
    void*              pNext;
    uint8_t            deviceUUID[VK_UUID_SIZE];
    uint8_t            driverUUID[VK_UUID_SIZE];
    uint8_t            deviceLUID[VK_LUID_SIZE];
    uint32_t           deviceNodeMask;
    VkBool32           deviceLUIDValid;
} VkPhysicalDeviceIDProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `deviceUUID` is an array of `VK_UUID_SIZE` `uint8_t` values representing a universally unique identifier for the device.
- `driverUUID` is an array of `VK_UUID_SIZE` `uint8_t` values representing a universally unique identifier for the driver build in use by the device.
- `deviceLUID` is an array of `VK_LUID_SIZE` `uint8_t` values representing a locally unique identifier for the device.
- `deviceNodeMask` is a `uint32_t` bitfield identifying the node within a linked device adapter corresponding to the device.
- `deviceLUIDValid` is a boolean value that will be `VK_TRUE` if `deviceLUID` contains a valid LUID and `deviceNodeMask` contains a valid node mask, and `VK_FALSE` if they do not.

If the `VkPhysicalDeviceIDProperties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with

each corresponding implementation-dependent property.

deviceUUID **must** be immutable for a given device across instances, processes, driver APIs, driver versions, and system reboots.

Applications **can** compare the **driverUUID** value across instance and process boundaries, and **can** make similar queries in external APIs to determine whether they are capable of sharing memory objects and resources using them with the device.

deviceUUID and/or **driverUUID** **must** be used to determine whether a particular external object can be shared between driver components, where such a restriction exists as defined in the compatibility table for the particular object type:

- [External memory handle types compatibility](#)
- [External semaphore handle types compatibility](#)
- [External fence handle types compatibility](#)

If **deviceLUIDValid** is **VK_FALSE**, the values of **deviceLUID** and **deviceNodeMask** are undefined. If **deviceLUIDValid** is **VK_TRUE** and Vulkan is running on the Windows operating system, the contents of **deviceLUID** **can** be cast to an **LUID** object and **must** be equal to the locally unique identifier of a **IDXGIAdapter1** object that corresponds to **physicalDevice**. If **deviceLUIDValid** is **VK_TRUE**, **deviceNodeMask** **must** contain exactly one bit. If Vulkan is running on an operating system that supports the Direct3D 12 API and **physicalDevice** corresponds to an individual device in a linked device adapter, **deviceNodeMask** identifies the Direct3D 12 node corresponding to **physicalDevice**. Otherwise, **deviceNodeMask** **must** be 1.

NOTE

Although they have identical descriptions, [VkPhysicalDeviceIDProperties::deviceUUID](#) may differ from [VkPhysicalDeviceProperties2::properties.pipelineCacheUUID](#). The former is intended to identify and correlate devices across API and driver boundaries, while the latter is used to identify a compatible device and driver combination to use when serializing and deserializing pipeline state.

Implementations **should** return **deviceUUID** values which are likely to be unique even in the presence of multiple Vulkan implementations (such as a GPU driver and a software renderer; two drivers for different GPUs; or the same Vulkan driver running on two logically different devices).

Khronos' conformance testing is unable to guarantee that **deviceUUID** values are actually unique, so implementors **should** make their own best efforts to ensure this. In particular, hard-coded **deviceUUID** values, especially all-0 bits, **should** never be used.

A combination of values unique to the vendor, the driver, and the hardware environment can be used to provide a **deviceUUID** which is unique to a high degree of certainty. Some possible inputs to such a computation are:

- Information reported by [vkGetPhysicalDeviceProperties](#)
- PCI device ID (if defined)

- PCI bus ID, or similar system configuration information.
- Driver binary checksums.

NOTE

While `VkPhysicalDeviceIDProperties::deviceUUID` is specified to remain consistent across driver versions and system reboots, it is not intended to be usable as a serializable persistent identifier for a device. It may change when a device is physically added to, removed from, or moved to a different connector in a system while that system is powered down. Further, there is no reasonable way to verify with conformance testing that a given device retains the same UUID in a given system across all driver versions supported in that system. While implementations should make every effort to report consistent device UUIDs across driver versions, applications should avoid relying on the persistence of this value for uses other than identifying compatible devices for external object sharing purposes.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceIDProperties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ID_PROPERTIES`

`VK_UUID_SIZE` is the length in `uint8_t` values of an array containing a universally unique device or driver build identifier, as returned in `VkPhysicalDeviceIDProperties::deviceUUID` and `VkPhysicalDeviceIDProperties::driverUUID`.

```
#define VK_UUID_SIZE          16U
```

`VK_LUID_SIZE` is the length in `uint8_t` values of an array containing a locally unique device identifier, as returned in `VkPhysicalDeviceIDProperties::deviceLUID`.

```
#define VK_LUID_SIZE          8U
```

The `VkPhysicalDeviceDriverProperties` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceDriverProperties {
    VkStructureType    sType;
    void*              pNext;
    VkDriverId          driverID;
    char                driverName[VK_MAX_DRIVER_NAME_SIZE];
    char                driverInfo[VK_MAX_DRIVER_INFO_SIZE];
    VkConformanceVersion conformanceVersion;
} VkPhysicalDeviceDriverProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.

- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `driverID` is a unique identifier for the driver of the physical device.
- `driverName` is an array of `VK_MAX_DRIVER_NAME_SIZE` `char` containing a null-terminated UTF-8 string which is the name of the driver.
- `driverInfo` is an array of `VK_MAX_DRIVER_INFO_SIZE` `char` containing a null-terminated UTF-8 string with additional information about the driver.
- `conformanceVersion` is the latest version of the Vulkan conformance test that the implementor has successfully tested this driver against prior to release (see [VkConformanceVersion](#)).

If the `VkPhysicalDeviceDriverProperties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

These are properties of the driver corresponding to a physical device.

`driverID` **must** be immutable for a given driver across instances, processes, driver versions, and system reboots.

Valid Usage (Implicit)

- `VOID-VkPhysicalDeviceDriverProperties-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRIVER_PROPERTIES`

Khronos driver IDs which **may** be returned in `VkPhysicalDeviceDriverProperties::driverID` are:

```
// Provided by VK_VERSION_1_2
typedef enum VkDriverId {
    VK_DRIVER_ID_AMD_PROPRIETARY = 1,
    VK_DRIVER_ID_AMD_OPEN_SOURCE = 2,
    VK_DRIVER_ID_MESA_RADV = 3,
    VK_DRIVER_ID_NVIDIA_PROPRIETARY = 4,
    VK_DRIVER_ID_INTEL_PROPRIETARY_WINDOWS = 5,
    VK_DRIVER_ID_INTEL_OPEN_SOURCE_MESA = 6,
    VK_DRIVER_ID_IMAGINATION_PROPRIETARY = 7,
    VK_DRIVER_ID_QUALCOMM_PROPRIETARY = 8,
    VK_DRIVER_ID_ARM_PROPRIETARY = 9,
    VK_DRIVER_ID_GOOGLE_SWIFTSHADER = 10,
    VK_DRIVER_ID_GGP_PROPRIETARY = 11,
    VK_DRIVER_ID_BROADCOM_PROPRIETARY = 12,
    VK_DRIVER_ID_MESA_LLVMPIPE = 13,
    VK_DRIVER_ID_MOLTENVK = 14,
    VK_DRIVER_ID_COREAVI_PROPRIETARY = 15,
    VK_DRIVER_ID_JUICE_PROPRIETARY = 16,
    VK_DRIVER_ID_VERISILICON_PROPRIETARY = 17,
    VK_DRIVER_ID_MESA_TURNIP = 18,
    VK_DRIVER_ID_MESA_V3DV = 19,
    VK_DRIVER_ID_MESA_PANVK = 20,
    VK_DRIVER_ID_SAMSUNG_PROPRIETARY = 21,
    VK_DRIVER_ID_MESA_VENUS = 22,
    VK_DRIVER_ID_MESA_DOZEN = 23,
    VK_DRIVER_ID_MESA_NVK = 24,
    VK_DRIVER_ID_IMAGINATION_OPEN_SOURCE_MESA = 25,
    VK_DRIVER_ID_MESA_HONEYKRISP = 26,
    VK_DRIVER_ID_VULKAN_SC_EMULATION_ON_VULKAN = 27,
    VK_DRIVER_ID_MESA_KOSMICKRISP = 28,
} VkDriverId;
```

NOTE

Khronos driver IDs may be allocated by vendors at any time. There may be multiple driver IDs for the same vendor, representing different drivers (for e.g. different platforms, proprietary or open source, etc.). Only the latest canonical versions of this Specification, of the corresponding [vk.xml](#) API Registry, and of the corresponding [vulkan_sc_core.h](#) header file **must** contain all reserved Khronos driver IDs.

Only driver IDs registered with Khronos are given symbolic names. There **may** be unregistered driver IDs returned.

[VK_MAX_DRIVER_NAME_SIZE](#) is the length in [char](#) values of an array containing a driver name string, as returned in [VkPhysicalDeviceDriverProperties::driverName](#).

```
#define VK_MAX_DRIVER_NAME_SIZE 256U
```

[VK_MAX_DRIVER_INFO_SIZE](#) is the length in [char](#) values of an array containing a driver information

string, as returned in [VkPhysicalDeviceDriverProperties::driverInfo](#).

```
#define VK_MAX_DRIVER_INFO_SIZE 256U
```

The conformance test suite version an implementation is compliant with is described with the [VkConformanceVersion](#) structure:

```
// Provided by VK_VERSION_1_2
typedef struct VkConformanceVersion {
    uint8_t    major;
    uint8_t    minor;
    uint8_t    subminor;
    uint8_t    patch;
} VkConformanceVersion;
```

- [major](#) is the major version number of the conformance test suite.
- [minor](#) is the minor version number of the conformance test suite.
- [subminor](#) is the subminor version number of the conformance test suite.
- [patch](#) is the patch version number of the conformance test suite.

The [VkPhysicalDevicePCIBusInfoPropertiesEXT](#) structure is defined as:

```
// Provided by VK_EXT_pci_bus_info
typedef struct VkPhysicalDevicePCIBusInfoPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t            pciDomain;
    uint32_t            pciBus;
    uint32_t            pciDevice;
    uint32_t            pciFunction;
} VkPhysicalDevicePCIBusInfoPropertiesEXT;
```

- [sType](#) is a [VkStructureType](#) value identifying this structure.
- [pNext](#) is [NULL](#) or a pointer to a structure extending this structure.
- [pciDomain](#) is the PCI bus domain.
- [pciBus](#) is the PCI bus identifier.
- [pciDevice](#) is the PCI device identifier.
- [pciFunction](#) is the PCI device function identifier.

If the [VkPhysicalDevicePCIBusInfoPropertiesEXT](#) structure is included in the [pNext](#) chain of the [VkPhysicalDeviceProperties2](#) structure passed to [vkGetPhysicalDeviceProperties2](#), it is filled in with each corresponding implementation-dependent property.

These are properties of the PCI bus information of a physical device.

Valid Usage (Implicit)

- VUID-VkPhysicalDevicePCIBusInfoPropertiesEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PCI_BUS_INFO_PROPERTIES_EXT`

To query properties of queues available on a physical device, call:

WARNING

This functionality is superseded by [Vulkan Version 1.1](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
void vkGetPhysicalDeviceQueueFamilyProperties(
    VkPhysicalDevice          physicalDevice,
    uint32_t*                 pQueueFamilyPropertyCount,
    VkQueueFamilyProperties*   pQueueFamilyProperties);
```

- `physicalDevice` is the handle to the physical device whose properties will be queried.
- `pQueueFamilyPropertyCount` is a pointer to an integer related to the number of queue families available or queried, as described below.
- `pQueueFamilyProperties` is either `NULL` or a pointer to an array of [VkQueueFamilyProperties](#) structures.

If `pQueueFamilyProperties` is `NULL`, then the number of queue families available is returned in `pQueueFamilyPropertyCount`. Implementations **must** support at least one queue family. Otherwise, `pQueueFamilyPropertyCount` **must** point to a variable set by the application to the number of elements in the `pQueueFamilyProperties` array, and on return the variable is overwritten with the number of structures actually written to `pQueueFamilyProperties`. If `pQueueFamilyPropertyCount` is less than the number of queue families available, at most `pQueueFamilyPropertyCount` structures will be written.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceQueueFamilyProperties-physicalDevice-parameter
`physicalDevice` **must** be a valid [VkPhysicalDevice](#) handle
- VUID-vkGetPhysicalDeviceQueueFamilyProperties-pQueueFamilyPropertyCount-parameter
`pQueueFamilyPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceQueueFamilyProperties-pQueueFamilyProperties-parameter
If the value referenced by `pQueueFamilyPropertyCount` is not `0`, and `pQueueFamilyProperties` is not `NULL`, `pQueueFamilyProperties` **must** be a valid pointer to an array of `pQueueFamilyPropertyCount` [VkQueueFamilyProperties](#) structures

The [VkQueueFamilyProperties](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkQueueFamilyProperties {
    VkQueueFlags    queueFlags;
    uint32_t        queueCount;
    uint32_t        timestampValidBits;
    VkExtent3D      minImageTransferGranularity;
} VkQueueFamilyProperties;
```

- **queueFlags** is a bitmask of [VkQueueFlagBits](#) indicating capabilities of the queues in this queue family.
- **queueCount** is the unsigned integer count of queues in this queue family. Each queue family **must** support at least one queue.
- **timestampValidBits** is the unsigned integer count of meaningful bits in the timestamps written via [vkCmdWriteTimestamp2KHR](#) or [vkCmdWriteTimestamp](#). The valid range for the count is 36 to 64 bits, or a value of 0, indicating no support for timestamps. Bits outside the valid range are guaranteed to be zeros.
- **minImageTransferGranularity** is the minimum granularity supported for image transfer operations on the queues in this queue family.

The value returned in **minImageTransferGranularity** has a unit of compressed texel blocks for images having a block-compressed format, and a unit of texels otherwise.

Possible values of **minImageTransferGranularity** are:

- (0,0,0) specifies that only whole mip levels **must** be transferred using the image transfer operations on the corresponding queues. In this case, the following restrictions apply to all offset and extent parameters of image transfer operations:
 - The **x**, **y**, and **z** members of a [VkOffset3D](#) parameter **must** always be zero.
 - The **width**, **height**, and **depth** members of a [VkExtent3D](#) parameter **must** always match the width, height, and depth of the image subresource corresponding to the parameter, respectively.
- (A_x , A_y , A_z) where A_x , A_y , and A_z are all integer powers of two. In this case the following restrictions apply to all image transfer operations:
 - **x**, **y**, and **z** of a [VkOffset3D](#) parameter **must** be integer multiples of A_x , A_y , and A_z , respectively.
 - **width** of a [VkExtent3D](#) parameter **must** be an integer multiple of A_x , or else **x** + **width** **must** equal the width of the image subresource corresponding to the parameter.
 - **height** of a [VkExtent3D](#) parameter **must** be an integer multiple of A_y , or else **y** + **height** **must** equal the height of the image subresource corresponding to the parameter.
 - **depth** of a [VkExtent3D](#) parameter **must** be an integer multiple of A_z , or else **z** + **depth** **must** equal the depth of the image subresource corresponding to the parameter.
 - If the format of the image corresponding to the parameters is one of the block-compressed formats then for the purposes of the above calculations the granularity **must** be scaled up

by the compressed texel block dimensions.

Queues supporting graphics and/or compute operations **must** report (1,1,1) in `minImageTransferGranularity`, meaning that there are no additional restrictions on the granularity of image transfer operations for these queues. Other queues supporting image transfer operations are only **required** to support whole mip level transfers, thus `minImageTransferGranularity` for queues belonging to such queue families **may** be (0,0,0).

The [Device Memory](#) section describes memory properties queried from the physical device.

For physical device feature queries see the [Features](#) chapter.

Bits which **may** be set in `VkQueueFamilyProperties::queueFlags`, indicating capabilities of queues in a queue family are:

```
// Provided by VK_VERSION_1_0
typedef enum VkQueueFlagBits {
    VK_QUEUE_GRAPHICS_BIT = 0x00000001,
    VK_QUEUE_COMPUTE_BIT = 0x00000002,
    VK_QUEUE_TRANSFER_BIT = 0x00000004,
    // Provided by VK_VERSION_1_1
    VK_QUEUE_PROTECTED_BIT = 0x00000010,
} VkQueueFlagBits;
```

- `VK_QUEUE_GRAPHICS_BIT` specifies that queues in this queue family support graphics operations.
- `VK_QUEUE_COMPUTE_BIT` specifies that queues in this queue family support compute operations.
- `VK_QUEUE_TRANSFER_BIT` specifies that queues in this queue family support transfer operations.
- `VK_QUEUE_SPARSE_BINDING_BIT` specifies that queues in this queue family support sparse memory management operations (see [Sparse Resources](#)). If any of the sparse resource features are enabled, then at least one queue family **must** support this bit. This flag is not supported in Vulkan SC [\[SCID-8\]](#).
- `VK_QUEUE_PROTECTED_BIT` specifies that queues in this queue family support the `VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT` bit. (see [Protected Memory](#)). If the physical device supports the `protectedMemory` feature, at least one of its queue families **must** support this bit.

If an implementation exposes any queue family that supports graphics operations, at least one queue family of at least one physical device exposed by the implementation **must** support both graphics and compute operations.

Furthermore, if the `protectedMemory` physical device feature is supported, then at least one queue family of at least one physical device exposed by the implementation **must** support graphics operations, compute operations, and protected memory operations.

NOTE

All commands that are allowed on a queue that supports transfer operations are also allowed on a queue that supports either graphics or compute operations. Thus, if the capabilities of a queue family include `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`, then reporting the `VK_QUEUE_TRANSFER_BIT` capability

separately for that queue family is **optional**.

For further details see [Queues](#).

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkQueueFlags;
```

VkQueueFlags is a bitmask type for setting a mask of zero or more [VkQueueFlagBits](#).

To query properties of queues available on a physical device, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceQueueFamilyProperties2(
    VkPhysicalDevice          physicalDevice,
    uint32_t*                 pQueueFamilyPropertyCount,
    VkQueueFamilyProperties2*  pQueueFamilyProperties);
```

- **physicalDevice** is the handle to the physical device whose properties will be queried.
- **pQueueFamilyPropertyCount** is a pointer to an integer related to the number of queue families available or queried, as described in [vkGetPhysicalDeviceQueueFamilyProperties](#).
- **pQueueFamilyProperties** is either **NULL** or a pointer to an array of [VkQueueFamilyProperties2](#) structures.

vkGetPhysicalDeviceQueueFamilyProperties2 behaves similarly to [vkGetPhysicalDeviceQueueFamilyProperties](#), with the ability to return extended information in a **pNext** chain of output structures.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceQueueFamilyProperties2-physicalDevice-parameter **physicalDevice** **must** be a valid [VkPhysicalDevice](#) handle
- VUID-vkGetPhysicalDeviceQueueFamilyProperties2-pQueueFamilyPropertyCount-parameter **pQueueFamilyPropertyCount** **must** be a valid pointer to a **uint32_t** value
- VUID-vkGetPhysicalDeviceQueueFamilyProperties2-pQueueFamilyProperties-parameter
If the value referenced by **pQueueFamilyPropertyCount** is not 0, and **pQueueFamilyProperties** is not **NULL**, **pQueueFamilyProperties** **must** be a valid pointer to an array of **pQueueFamilyPropertyCount** [VkQueueFamilyProperties2](#) structures

The **VkQueueFamilyProperties2** structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkQueueFamilyProperties2 {
    VkStructureType    sType;
    void*              pNext;
    VkQueueFamilyProperties queueFamilyProperties;
} VkQueueFamilyProperties2;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `queueFamilyProperties` is a `VkQueueFamilyProperties` structure which is populated with the same values as in `vkGetPhysicalDeviceQueueFamilyProperties`.

Valid Usage (Implicit)

- VUID-VkQueueFamilyProperties2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_QUEUE_FAMILY_PROPERTIES_2`
- VUID-VkQueueFamilyProperties2-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkQueueFamilyGlobalPriorityProperties`
- VUID-VkQueueFamilyProperties2-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique

The `VkQueueFamilyGlobalPriorityProperties` structure is defined as:

```
typedef struct VkQueueFamilyGlobalPriorityProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           priorityCount;
    VkQueueGlobalPriority priorities[VK_MAX_GLOBAL_PRIORITY_SIZE];
} VkQueueFamilyGlobalPriorityProperties;
```

```
// Provided by VK_KHR_global_priority
typedef VkQueueFamilyGlobalPriorityProperties
VkQueueFamilyGlobalPriorityPropertiesKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `priorityCount` is the number of supported global queue priorities in this queue family, and it **must** be greater than 0.
- `priorities` is an array of `VK_MAX_GLOBAL_PRIORITY_SIZE` `VkQueueGlobalPriority` enums representing all supported global queue priorities in this queue family. The first `priorityCount` elements of the array will be valid.

If the `VkQueueFamilyGlobalPriorityProperties` structure is included in the `pNext` chain of the `VkQueueFamilyProperties2` structure passed to `vkGetPhysicalDeviceQueueFamilyProperties2`, it is filled in with the list of supported global queue priorities for the indicated family.

The valid elements of `priorities` **must** not contain any duplicate values.

The valid elements of `priorities` **must** be a continuous sequence of `VkQueueGlobalPriority` enums in the ascending order.

NOTE

For example, returning `priorityCount` as 3 with supported `priorities` as `VK_QUEUE_GLOBAL_PRIORITY_LOW`, `VK_QUEUE_GLOBAL_PRIORITY_MEDIUM` and `VK_QUEUE_GLOBAL_PRIORITY_REALTIME` is not allowed.

Valid Usage (Implicit)

- `VOID-VkQueueFamilyGlobalPriorityProperties-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_QUEUE_FAMILY_GLOBAL_PRIORITY_PROPERTIES`

`VK_MAX_GLOBAL_PRIORITY_SIZE` is the length of an array of `VkQueueGlobalPriority` enumerants representing supported queue priorities, as returned in `VkQueueFamilyGlobalPriorityProperties::priorities`.

```
#define VK_MAX_GLOBAL_PRIORITY_SIZE 16U
```

```
#define VK_MAX_GLOBAL_PRIORITY_SIZE_KHR VK_MAX_GLOBAL_PRIORITY_SIZE
```

To enumerate the performance query counters available on a queue family of a physical device, call:

```
// Provided by VK_KHR_performance_query
VkResult vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR(
    VkPhysicalDevice          physicalDevice,
    uint32_t                  queueFamilyIndex,
    uint32_t*                  pCounterCount,
    VkPerformanceCounterKHR*   pCounters,
    VkPerformanceCounterDescriptionKHR* pCounterDescriptions);
```

- `physicalDevice` is the handle to the physical device whose queue family performance query counter properties will be queried.
- `queueFamilyIndex` is the index into the queue family of the physical device we want to get properties for.
- `pCounterCount` is a pointer to an integer related to the number of counters available or queried, as described below.

- `pCounters` is either `NULL` or a pointer to an array of `VkPerformanceCounterKHR` structures.
- `pCounterDescriptions` is either `NULL` or a pointer to an array of `VkPerformanceCounterDescriptionKHR` structures.

If `pCounters` is `NULL` and `pCounterDescriptions` is `NULL`, then the number of counters available is returned in `pCounterCount`. Otherwise, `pCounterCount` **must** point to a variable set by the application to the number of elements in the `pCounters`, `pCounterDescriptions`, or both arrays and on return the variable is overwritten with the number of structures actually written out. If `pCounterCount` is less than the number of counters available, at most `pCounterCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available counters were returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR-physicalDevice-parameter
`physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR-pCounterCount-parameter
`pCounterCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR-pCounters-parameter
If the value referenced by `pCounterCount` is not `0`, and `pCounters` is not `NULL`, `pCounters` **must** be a valid pointer to an array of `pCounterCount` `VkPerformanceCounterKHR` structures
- VUID-vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR-pCounterDescriptions-parameter
If the value referenced by `pCounterCount` is not `0`, and `pCounterDescriptions` is not `NULL`, `pCounterDescriptions` **must** be a valid pointer to an array of `pCounterCount` `VkPerformanceCounterDescriptionKHR` structures

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkPerformanceCounterKHR` structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkPerformanceCounterKHR {
    VkStructureType      sType;
    void*                pNext;
    VkPerformanceCounterUnitKHR    unit;
    VkPerformanceCounterScopeKHR   scope;
    VkPerformanceCounterStorageKHR storage;
    uint8_t              uuid[VK_UUID_SIZE];
} VkPerformanceCounterKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `unit` is a `VkPerformanceCounterUnitKHR` specifying the unit that the counter data will record.
- `scope` is a `VkPerformanceCounterScopeKHR` specifying the scope that the counter belongs to.
- `storage` is a `VkPerformanceCounterStorageKHR` specifying the storage type that the counter's data uses.
- `uuid` is an array of size `VK_UUID_SIZE`, containing 8-bit values that represent a universally unique identifier for the counter of the physical device.

Valid Usage (Implicit)

- VUID-VkPerformanceCounterKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PERFORMANCE_COUNTER_KHR`
- VUID-VkPerformanceCounterKHR-pNext-pNext
`pNext` **must** be `NULL`

Performance counters have an associated unit. This unit describes how to interpret the performance counter result.

The performance counter unit types which **may** be returned in `VkPerformanceCounterKHR::unit` are:

```
// Provided by VK_KHR_performance_query
typedef enum VkPerformanceCounterUnitKHR {
    VK_PERFORMANCE_COUNTER_UNIT_GENERIC_KHR = 0,
    VK_PERFORMANCE_COUNTER_UNIT_PERCENTAGE_KHR = 1,
    VK_PERFORMANCE_COUNTER_UNIT_NANOSECONDS_KHR = 2,
    VK_PERFORMANCE_COUNTER_UNIT_BYTES_KHR = 3,
    VK_PERFORMANCE_COUNTER_UNIT_BYTES_PER_SECOND_KHR = 4,
    VK_PERFORMANCE_COUNTER_UNIT_KELVIN_KHR = 5,
    VK_PERFORMANCE_COUNTER_UNIT_WATTS_KHR = 6,
    VK_PERFORMANCE_COUNTER_UNIT_VOLTS_KHR = 7,
    VK_PERFORMANCE_COUNTER_UNIT_AMPS_KHR = 8,
    VK_PERFORMANCE_COUNTER_UNIT_HERTZ_KHR = 9,
    VK_PERFORMANCE_COUNTER_UNIT_CYCLES_KHR = 10,
} VkPerformanceCounterUnitKHR;
```

- `VK_PERFORMANCE_COUNTER_UNIT_GENERIC_KHR` - the performance counter unit is a generic data point.
- `VK_PERFORMANCE_COUNTER_UNIT_PERCENTAGE_KHR` - the performance counter unit is a percentage (%).
- `VK_PERFORMANCE_COUNTER_UNIT_NANOSECONDS_KHR` - the performance counter unit is a value of nanoseconds (ns).
- `VK_PERFORMANCE_COUNTER_UNIT_BYTES_KHR` - the performance counter unit is a value of bytes.
- `VK_PERFORMANCE_COUNTER_UNIT_BYTES_PER_SECOND_KHR` - the performance counter unit is a value of bytes/s.
- `VK_PERFORMANCE_COUNTER_UNIT_KELVIN_KHR` - the performance counter unit is a temperature reported in Kelvin.
- `VK_PERFORMANCE_COUNTER_UNIT_WATTS_KHR` - the performance counter unit is a value of watts (W).
- `VK_PERFORMANCE_COUNTER_UNIT_VOLTS_KHR` - the performance counter unit is a value of volts (V).
- `VK_PERFORMANCE_COUNTER_UNIT_AMPS_KHR` - the performance counter unit is a value of amps (A).
- `VK_PERFORMANCE_COUNTER_UNIT_HERTZ_KHR` - the performance counter unit is a value of hertz (Hz).
- `VK_PERFORMANCE_COUNTER_UNIT_CYCLES_KHR` - the performance counter unit is a value of cycles.

Performance counters have an associated scope. This scope describes the granularity of a performance counter.

The performance counter scope types which **may** be returned in `VkPerformanceCounterKHR::scope` are:

```
// Provided by VK_KHR_performance_query
typedef enum VkPerformanceCounterScopeKHR {
    VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_BUFFER_KHR = 0,
    VK_PERFORMANCE_COUNTER_SCOPE_RENDER_PASS_KHR = 1,
    VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_KHR = 2,
    // VK_QUERY_SCOPE_COMMAND_BUFFER_KHR is a legacy alias
    VK_QUERY_SCOPE_COMMAND_BUFFER_KHR =
VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_BUFFER_KHR,
    // VK_QUERY_SCOPE_RENDER_PASS_KHR is a legacy alias
    VK_QUERY_SCOPE_RENDER_PASS_KHR = VK_PERFORMANCE_COUNTER_SCOPE_RENDER_PASS_KHR,
    // VK_QUERY_SCOPE_COMMAND_KHR is a legacy alias
    VK_QUERY_SCOPE_COMMAND_KHR = VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_KHR,
} VkPerformanceCounterScopeKHR;
```

- **VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_BUFFER_KHR** - the performance counter scope is a single complete command buffer.
- **VK_PERFORMANCE_COUNTER_SCOPE_RENDER_PASS_KHR** - the performance counter scope is zero or more complete render passes. The performance query containing the performance counter **must** begin and end outside a render pass instance.
- **VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_KHR** - the performance counter scope is zero or more commands.

Performance counters have an associated storage. This storage describes the payload of a counter result.

The performance counter storage types which **may** be returned in [VkPerformanceCounterKHR::storage](#) are:

```
// Provided by VK_KHR_performance_query
typedef enum VkPerformanceCounterStorageKHR {
    VK_PERFORMANCE_COUNTER_STORAGE_INT32_KHR = 0,
    VK_PERFORMANCE_COUNTER_STORAGE_INT64_KHR = 1,
    VK_PERFORMANCE_COUNTER_STORAGE_UINT32_KHR = 2,
    VK_PERFORMANCE_COUNTER_STORAGE_UINT64_KHR = 3,
    VK_PERFORMANCE_COUNTER_STORAGE_FLOAT32_KHR = 4,
    VK_PERFORMANCE_COUNTER_STORAGE_FLOAT64_KHR = 5,
} VkPerformanceCounterStorageKHR;
```

- **VK_PERFORMANCE_COUNTER_STORAGE_INT32_KHR** - the performance counter storage is a 32-bit signed integer.
- **VK_PERFORMANCE_COUNTER_STORAGE_INT64_KHR** - the performance counter storage is a 64-bit signed integer.
- **VK_PERFORMANCE_COUNTER_STORAGE_UINT32_KHR** - the performance counter storage is a 32-bit unsigned integer.
- **VK_PERFORMANCE_COUNTER_STORAGE_UINT64_KHR** - the performance counter storage is a 64-bit unsigned integer.

- `VK_PERFORMANCE_COUNTER_STORAGE_FLOAT32_KHR` - the performance counter storage is a 32-bit floating-point.
- `VK_PERFORMANCE_COUNTER_STORAGE_FLOAT64_KHR` - the performance counter storage is a 64-bit floating-point.

The `VkPerformanceCounterDescriptionKHR` structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkPerformanceCounterDescriptionKHR {
    VkStructureType           sType;
    void*                     pNext;
    VkPerformanceCounterDescriptionFlagsKHR flags;
    char                      name[VK_MAX_DESCRIPTION_SIZE];
    char                      category[VK_MAX_DESCRIPTION_SIZE];
    char                      description[VK_MAX_DESCRIPTION_SIZE];
} VkPerformanceCounterDescriptionKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkPerformanceCounterDescriptionFlagBitsKHR` indicating the usage behavior for the counter.
- `name` is an array of size `VK_MAX_DESCRIPTION_SIZE`, containing a null-terminated UTF-8 string specifying the name of the counter.
- `category` is an array of size `VK_MAX_DESCRIPTION_SIZE`, containing a null-terminated UTF-8 string specifying the category of the counter.
- `description` is an array of size `VK_MAX_DESCRIPTION_SIZE`, containing a null-terminated UTF-8 string specifying the description of the counter.

Valid Usage (Implicit)

- VUID-VkPerformanceCounterDescriptionKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PERFORMANCE_COUNTER_DESCRIPTION_KHR`
- VUID-VkPerformanceCounterDescriptionKHR-pNext-pNext
`pNext` **must** be `NULL`

Bits which **can** be set in `VkPerformanceCounterDescriptionKHR::flags`, specifying usage behavior of a performance counter, are:

```
// Provided by VK_KHR_performance_query
typedef enum VkPerformanceCounterDescriptionFlagBitsKHR {
    VK_PERFORMANCE_COUNTER_DESCRIPTION_PERFORMANCE_IMPACTING_BIT_KHR = 0x00000001,
    VK_PERFORMANCE_COUNTER_DESCRIPTION_CONCURRENTLY_IMPACTED_BIT_KHR = 0x00000002,
    // VK_PERFORMANCE_COUNTER_DESCRIPTION_PERFORMANCE_IMPACTING_KHR is a legacy alias
    VK_PERFORMANCE_COUNTER_DESCRIPTION_PERFORMANCE_IMPACTING_KHR =
VK_PERFORMANCE_COUNTER_DESCRIPTION_PERFORMANCE_IMPACTING_BIT_KHR,
    // VK_PERFORMANCE_COUNTER_DESCRIPTION_CONCURRENTLY_IMPACTED_KHR is a legacy alias
    VK_PERFORMANCE_COUNTER_DESCRIPTION_CONCURRENTLY_IMPACTED_KHR =
VK_PERFORMANCE_COUNTER_DESCRIPTION_CONCURRENTLY_IMPACTED_BIT_KHR,
} VkPerformanceCounterDescriptionFlagBitsKHR;
```

- `VK_PERFORMANCE_COUNTER_DESCRIPTION_PERFORMANCE_IMPACTING_BIT_KHR` specifies that recording the counter **may** have a noticeable performance impact.
- `VK_PERFORMANCE_COUNTER_DESCRIPTION_CONCURRENTLY_IMPACTED_BIT_KHR` specifies that concurrently recording the counter while other submitted command buffers are running **may** impact the accuracy of the recording.

```
// Provided by VK_KHR_performance_query
typedef VkFlags VkPerformanceCounterDescriptionFlagsKHR;
```

`VkPerformanceCounterDescriptionFlagsKHR` is a bitmask type for setting a mask of zero or more `VkPerformanceCounterDescriptionFlagBitsKHR`.

5.2. Devices

Device objects represent logical connections to physical devices. Each device exposes a number of *queue families* each having one or more *queues*. All queues in a queue family support the same operations.

As described in [Physical Devices](#), a Vulkan application will first query for all physical devices in a system. Each physical device **can** then be queried for its capabilities, including its queue and queue family properties. Once an acceptable physical device is identified, an application will create a corresponding logical device. The created logical device is then the primary interface to the physical device.

How to enumerate the physical devices in a system and query those physical devices for their queue family properties is described in the [Physical Device Enumeration](#) section above.

A single logical device **can** be created from multiple physical devices, if those physical devices belong to the same device group. A *device group* is a set of physical devices that support accessing each other's memory and recording a single command buffer that **can** be executed on all the physical devices. Device groups are enumerated by calling `vkEnumeratePhysicalDeviceGroups`, and a logical device is created from a subset of the physical devices in a device group by passing the physical devices through `VkDeviceGroupDeviceCreateInfo`. For two physical devices to be in the same device group, they **must** support identical extensions, features, and properties.

NOTE

Physical devices in the same device group **must** be so similar because there are no rules for how different features/properties would interact. They **must** return the same values for nearly every invariant `vkGetPhysicalDevice*` feature, property, capability, etc., but could potentially differ for certain queries based on things like having a different display connected, or a different compositor. The specification does not attempt to enumerate which state is in each category, because such a list would quickly become out of date.

To retrieve a list of the device groups present in the system, call:

```
// Provided by VK_VERSION_1_1
VkResult vkEnumeratePhysicalDeviceGroups(
    VkInstance          instance,
    uint32_t*           pPhysicalDeviceGroupCount,
    VkPhysicalDeviceGroupProperties* pPhysicalDeviceGroupProperties);
```

- `instance` is a handle to a Vulkan instance previously created with `vkCreateInstance`.
- `pPhysicalDeviceGroupCount` is a pointer to an integer related to the number of device groups available or queried, as described below.
- `pPhysicalDeviceGroupProperties` is either `NULL` or a pointer to an array of `VkPhysicalDeviceGroupProperties` structures.

If `pPhysicalDeviceGroupProperties` is `NULL`, then the number of device groups available is returned in `pPhysicalDeviceGroupCount`. Otherwise, `pPhysicalDeviceGroupCount` **must** point to a variable set by the application to the number of elements in the `pPhysicalDeviceGroupProperties` array, and on return the variable is overwritten with the number of structures actually written to `pPhysicalDeviceGroupProperties`. If `pPhysicalDeviceGroupCount` is less than the number of device groups available, at most `pPhysicalDeviceGroupCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available device groups were returned.

Every physical device **must** be in exactly one device group.

Valid Usage (Implicit)

- VUID-vkEnumeratePhysicalDeviceGroups-instance-parameter
`instance` **must** be a valid `VkInstance` handle
- VUID-vkEnumeratePhysicalDeviceGroups-pPhysicalDeviceGroupCount-parameter
`pPhysicalDeviceGroupCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkEnumeratePhysicalDeviceGroups-pPhysicalDeviceGroupProperties-parameter
If the value referenced by `pPhysicalDeviceGroupCount` is not `0`, and `pPhysicalDeviceGroupProperties` is not `NULL`, `pPhysicalDeviceGroupProperties` **must** be a valid pointer to an array of `pPhysicalDeviceGroupCount` `VkPhysicalDeviceGroupProperties` structures

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkPhysicalDeviceGroupProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceGroupProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           physicalDeviceCount;
    VkPhysicalDevice    physicalDevices[VK_MAX_DEVICE_GROUP_SIZE];
    VkBool32           subsetAllocation;
} VkPhysicalDeviceGroupProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `physicalDeviceCount` is the number of physical devices in the group.
- `physicalDevices` is an array of `VK_MAX_DEVICE_GROUP_SIZE` `VkPhysicalDevice` handles representing all physical devices in the group. The first `physicalDeviceCount` elements of the array will be valid.
- `subsetAllocation` specifies whether logical devices created from the group support allocating device memory on a subset of devices, via the `deviceMask` member of the `VkMemoryAllocateFlagsInfo`. If this is `VK_FALSE`, then all device memory allocations are made across all physical devices in the group. If `physicalDeviceCount` is 1, then `subsetAllocation` **must** be `VK_FALSE`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceGroupProperties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GROUP_PROPERTIES`
- VUID-VkPhysicalDeviceGroupProperties-pNext-pNext
`pNext` **must** be `NULL`

`VK_MAX_DEVICE_GROUP_SIZE` is the length of an array containing `VkPhysicalDevice` handle values representing all physical devices in a group, as returned in `VkPhysicalDeviceGroupProperties::physicalDevices`.

```
#define VK_MAX_DEVICE_GROUP_SIZE 32U
```

5.2.1. Device Creation

Logical devices are represented by `VkDevice` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_HANDLE(VkDevice)
```

A logical device is created as a *connection* to a physical device. To create a logical device, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateDevice(
    VkPhysicalDevice          physicalDevice,
    const VkDeviceCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator,
    VkDevice*                 pDevice);
```

- `physicalDevice` **must** be one of the device handles returned from a call to `vkEnumeratePhysicalDevices` (see [Physical Device Enumeration](#)).
- `pCreateInfo` is a pointer to a `VkDeviceCreateInfo` structure containing information about how to create the device.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pDevice` is a pointer to a handle in which the created `VkDevice` is returned.

`vkCreateDevice` verifies that extensions and features requested in the `ppEnabledExtensionNames` and `pEnabledFeatures` members of `pCreateInfo`, respectively, are supported by the implementation. If any requested extension is not supported, `vkCreateDevice` **must** return `VK_ERROR_EXTENSION_NOT_PRESENT`. If any requested feature is not supported, `vkCreateDevice` **must** return `VK_ERROR_FEATURE_NOT_PRESENT`. Support for extensions **can** be checked before creating a device by querying `vkEnumerateDeviceExtensionProperties`. Support for features **can** similarly be checked by querying `vkGetPhysicalDeviceFeatures`.

`vkCreateDevice` also verifies that mandatory structures and features for Vulkan SC are present and enabled:

- The `pNext` chain **must** include a `VkDeviceObjectReservationCreateInfo` structure.
- The `pNext` chain **must** include a `VkPhysicalDeviceVulkanSC10Features` structure.

If any of these conditions are not met, `vkCreateDevice` **must** return `VK_ERROR_INITIALIZATION_FAILED`.

After verifying and enabling the extensions the `VkDevice` object is created and returned to the application.

An implementation **may** allow multiple logical devices to be created from the same physical device. Logical device creation **may** fail due to lack of device-specific resources, including too many other logical devices, in addition to other errors. If that occurs, `vkCreateDevice` will return `VK_ERROR_TOO_MANY_OBJECTS`.

If the pipeline cache data pointed to by the `pInitialData` member of any element of `VkDeviceObjectReservationCreateInfo::pPipelineCacheCreateInfos` is not compatible with the device, then `vkCreateDevice` will return `VK_ERROR_INVALID_PIPELINE_CACHE_DATA`.

Some implementations **may** only accept pipeline cache data pointers that conform to platform and/or implementation specific alignment requirements. These alignment requirements cannot exceed 8 bytes. If any of the pipeline cache data pointers specified in the `pInitialData` member of the elements of `VkDeviceObjectReservationCreateInfo::pPipelineCacheCreateInfos` are not aligned according to these requirements, then `vkCreateDevice` will return `VK_ERROR_INVALID_PIPELINE_CACHE_DATA`.

To provide *application parameters* at device creation time, an application **can** link one or more `VkApplicationParametersEXT` structures to the `pNext` chain of the `VkDeviceCreateInfo` structure.

If the `VkApplicationParametersEXT::vendorID` and `VkApplicationParametersEXT::deviceID` values do not match the `VkPhysicalDeviceProperties::vendorID` and `VkPhysicalDeviceProperties::deviceID` of `physicalDevice`, `vkCreateDevice` **must** return `VK_ERROR_INITIALIZATION_FAILED`.

If `VkApplicationParametersEXT::key` is not a valid implementation-defined application parameter key for the device being created, or if `value` is not a valid value for the specified `key`, `vkCreateDevice` will fail and return `VK_ERROR_INITIALIZATION_FAILED`.

For any implementation-defined application parameter `key` that exists but is not set by the application, the implementation-specific default value is used.

Valid Usage

- VUID-vkCreateDevice-ppEnabledExtensionNames-01387
All **required device extensions** for each extension in the `VkDeviceCreateInfo::ppEnabledExtensionNames` list **must** also be present in that list.
- VUID-vkCreateDevice-key-05092
The `key` value of each `VkApplicationParametersEXT` structure in the `VkDeviceCreateInfo::pNext` chain **must** be unique
- VUID-vkCreateDevice-deviceMemoryRequestCount-05095
The sum of `deviceMemoryRequestCount` over all `VkDeviceObjectReservationCreateInfo` structures included in the `VkDeviceCreateInfo::pNext` chain **must** be less than or equal to `VkPhysicalDeviceLimits::maxMemoryAllocationCount`
- VUID-vkCreateDevice-samplerRequestCount-05096
The sum of `samplerRequestCount` over all `VkDeviceObjectReservationCreateInfo` structures included in the `VkDeviceCreateInfo::pNext` chain **must** be less than or equal to

Valid Usage (Implicit)

- VUID-vkCreateDevice-physicalDevice-parameter
physicalDevice must be a valid [VkPhysicalDevice](#) handle
- VUID-vkCreateDevice-pCreateInfo-parameter
pCreateInfo must be a valid pointer to a valid [VkDeviceCreateInfo](#) structure
- VUID-vkCreateDevice-pAllocator-null
pAllocator must be **NULL**
- VUID-vkCreateDevice-pDevice-parameter
pDevice must be a valid pointer to a [VkDevice](#) handle

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_DEVICE_LOST**
- **VK_ERROR_EXTENSION_NOT_PRESENT**
- **VK_ERROR_FEATURE_NOT_PRESENT**
- **VK_ERROR_INITIALIZATION_FAILED**
- **VK_ERROR_INVALID_PIPELINE_CACHE_DATA**
- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_TOO_MANY_OBJECTS**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The [VkDeviceCreateInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDeviceCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkDeviceCreateFlags       flags;
    uint32_t                  queueCreateInfoCount;
    const VkDeviceQueueCreateInfo* pQueueCreateInfos;
    // enabledLayerCount is legacy and should not be used
    uint32_t                  enabledLayerCount;
    // ppEnabledLayerNames is legacy and should not be used
    const char* const*        ppEnabledLayerNames;
    uint32_t                  enabledExtensionCount;
    const char* const*        ppEnabledExtensionNames;
    const VkPhysicalDeviceFeatures* pEnabledFeatures;
} VkDeviceCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `queueCreateInfoCount` is the unsigned integer size of the `pQueueCreateInfos` array. Refer to the [Queue Creation](#) section below for further details.
- `pQueueCreateInfos` is a pointer to an array of `VkDeviceQueueCreateInfo` structures describing the queues that are requested to be created along with the logical device. Refer to the [Queue Creation](#) section below for further details.
- `enabledLayerCount` is legacy and ignored. See [Device Layers: Superseded via instance layers](#).
- `ppEnabledLayerNames` is legacy and ignored. See [Device Layers: Superseded via instance layers](#).
- `enabledExtensionCount` is the number of device extensions to enable.
- `ppEnabledExtensionNames` is a pointer to an array of `enabledExtensionCount` null-terminated UTF-8 strings containing the names of extensions to enable for the created device. See the [Extensions](#) section for further details.
- `pEnabledFeatures` is `NULL` or a pointer to a `VkPhysicalDeviceFeatures` structure containing boolean indicators of all the features to be enabled. Refer to the [Features](#) section for further details. This field is legacy. See [Physical Device Queries: Superseded via version 2](#).

Valid Usage

- VUID-VkDeviceCreateInfo-queueFamilyIndex-02802
The `queueFamilyIndex` member of each element of `pQueueCreateInfos` **must** be unique within `pQueueCreateInfos`, except that two members can share the same `queueFamilyIndex` if one describes protected-capable queues and one describes queues that are not protected-capable
- VUID-VkDeviceCreateInfo-pQueueCreateInfos-06755
If multiple elements of `pQueueCreateInfos` share the same `queueFamilyIndex`, the sum of

their `queueCount` members **must** be less than or equal to the `queueCount` member of the `VkQueueFamilyProperties` structure, as returned by `vkGetPhysicalDeviceQueueFamilyProperties` in the `pQueueFamilyProperties[queueFamilyIndex]`

- VUID-VkDeviceCreateInfo-pQueueCreateInfos-06654

If multiple elements of `pQueueCreateInfos` share the same `queueFamilyIndex`, then all of such elements **must** have the same global priority level, which **can** be specified explicitly by the including a `VkDeviceQueueGlobalPriorityCreateInfo` structure in the `pNext` chain, or by the implicit default value

- VUID-VkDeviceCreateInfo-pNext-00373

If the `pNext` chain includes a `VkPhysicalDeviceFeatures2` structure, then `pEnabledFeatures` **must** be `NULL`

- VUID-VkDeviceCreateInfo-pNext-02829

If the `pNext` chain includes a `VkPhysicalDeviceVulkan11Features` structure, then it **must** not include a `VkPhysicalDevice16BitStorageFeatures`, `VkPhysicalDeviceMultiviewFeatures`, `VkPhysicalDeviceVariablePointersFeatures`, `VkPhysicalDeviceProtectedMemoryFeatures`, `VkPhysicalDeviceSamplerYcbcrConversionFeatures`, or `VkPhysicalDeviceShaderDrawParametersFeatures` structure

- VUID-VkDeviceCreateInfo-pNext-02830

If the `pNext` chain includes a `VkPhysicalDeviceVulkan12Features` structure, then it **must** not include a `VkPhysicalDevice8BitStorageFeatures`, `VkPhysicalDeviceShaderAtomicInt64Features`, `VkPhysicalDeviceShaderFloat16Int8Features`, `VkPhysicalDeviceDescriptorIndexingFeatures`, `VkPhysicalDeviceScalarBlockLayoutFeatures`, `VkPhysicalDeviceImagelessFramebufferFeatures`, `VkPhysicalDeviceUniformBufferStandardLayoutFeatures`, `VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures`, `VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures`, `VkPhysicalDeviceHostQueryResetFeatures`, `VkPhysicalDeviceTimelineSemaphoreFeatures`, `VkPhysicalDeviceBufferDeviceAddressFeatures`, or `VkPhysicalDeviceVulkanMemoryModelFeatures` structure

- VUID-VkDeviceCreateInfo-None-04896

If the `sparseImageInt64Atomics` feature is enabled, `shaderImageInt64Atomics` **must** be enabled

- VUID-VkDeviceCreateInfo-None-04897

If the `sparseImageFloat32Atomics` feature is enabled, `shaderImageFloat32Atomics` **must** be enabled

- VUID-VkDeviceCreateInfo-None-04898

If the `sparseImageFloat32AtomicAdd` feature is enabled, `shaderImageFloat32AtomicAdd` **must** be enabled

- VUID-VkDeviceCreateInfo-robustBufferAccess-10247

If the `robustBufferAccess` feature is enabled, and `robustBufferAccessUpdateAfterBind` is

`VK_FALSE`, then `descriptorBindingUniformBufferUpdateAfterBind`, `descriptorBindingStorageBufferUpdateAfterBind`, `descriptorBindingUniformTexelBufferUpdateAfterBind`, and `descriptorBindingStorageTexelBufferUpdateAfterBind` **must** not be enabled

- `VUID-VkDeviceCreateInfo-None-10778`
`queueCreateInfoCount` **must** be greater than 0

Valid Usage (Implicit)

- `VUID-VkDeviceCreateInfo-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO`
- `VUID-VkDeviceCreateInfo-pNext-pNext`
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkApplicationParametersEXT`, `VkDeviceGroupDeviceCreateInfo`, `VkDeviceObjectReservationCreateInfo`, `VkDeviceSemaphoreSciSyncPoolReservationCreateInfoNV`, `VkFaultCallbackInfo`, `VkPerformanceQueryReservationInfoKHR`, `VkPhysicalDevice16BitStorageFeatures`, `VkPhysicalDevice4444FormatsFeaturesEXT`, `VkPhysicalDevice8BitStorageFeatures`, `VkPhysicalDeviceASTCDecodeFeaturesEXT`, `VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT`, `VkPhysicalDeviceBufferDeviceAddressFeatures`, `VkPhysicalDeviceColorWriteEnableFeaturesEXT`, `VkPhysicalDeviceCustomBorderColorFeaturesEXT`, `VkPhysicalDeviceDepthClipEnableFeaturesEXT`, `VkPhysicalDeviceDescriptorIndexingFeatures`, `VkPhysicalDeviceExtendedDynamicState2FeaturesEXT`, `VkPhysicalDeviceExtendedDynamicStateFeaturesEXT`, `VkPhysicalDeviceExternalMemorySciBufFeaturesNV`, `VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX`, `VkPhysicalDeviceExternalSciSync2FeaturesNV`, `VkPhysicalDeviceExternalSciSyncFeaturesNV`, `VkPhysicalDeviceFeatures2`, `VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT`, `VkPhysicalDeviceFragmentShadingRateFeaturesKHR`, `VkPhysicalDeviceGlobalPriorityQueryFeatures`, `VkPhysicalDeviceHostQueryResetFeatures`, `VkPhysicalDeviceImageRobustnessFeatures`, `VkPhysicalDeviceImagelessFramebufferFeatures`, `VkPhysicalDeviceIndexTypeUint8Features`, `VkPhysicalDeviceLineRasterizationFeatures`, `VkPhysicalDeviceMultiviewFeatures`, `VkPhysicalDevicePerformanceQueryFeaturesKHR`, `VkPhysicalDeviceProtectedMemoryFeatures`, `VkPhysicalDeviceRobustness2FeaturesKHR`, `VkPhysicalDeviceSamplerYcbcrConversionFeatures`, `VkPhysicalDeviceScalarBlockLayoutFeatures`, `VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures`, `VkPhysicalDeviceShaderAtomicFloatFeaturesEXT`, `VkPhysicalDeviceShaderAtomicInt64Features`, `VkPhysicalDeviceShaderClockFeaturesKHR`, `VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures`,

VkPhysicalDeviceShaderDrawParametersFeatures,
 VkPhysicalDeviceShaderFloat16Int8Features,
 VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT,
 VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures,
 VkPhysicalDeviceShaderTerminateInvocationFeatures,
 VkPhysicalDeviceSubgroupSizeControlFeatures,
 VkPhysicalDeviceSynchronization2Features,
 VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT,
 VkPhysicalDeviceTextureCompressionASTCHDRFeatures,
 VkPhysicalDeviceTimelineSemaphoreFeatures,
 VkPhysicalDeviceUniformBufferStandardLayoutFeatures,
 VkPhysicalDeviceVariablePointersFeatures,
 VkPhysicalDeviceVertexAttributeDivisorFeatures,
 VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT,
 VkPhysicalDeviceVulkan11Features, VkPhysicalDeviceVulkan12Features,
 VkPhysicalDeviceVulkanMemoryModelFeatures, VkPhysicalDeviceVulkanSC10Features,
 VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT, or
 VkPhysicalDeviceYcbcrImageArraysFeaturesEXT

- VUID-VkDeviceCreateInfo-sType-unique
The **sType** value of each structure in the **pNext** chain **must** be unique, with the exception of structures of type [VkApplicationParametersEXT](#), [VkDeviceObjectReservationCreateInfo](#), [VkDeviceSemaphoreSciSyncPoolReservationCreateInfoNV](#), or [VkPerformanceQueryReservationInfoKHR](#)
- VUID-VkDeviceCreateInfo-flags-zero-bitmask
flags **must** be 0
- VUID-VkDeviceCreateInfo-pQueueCreateInfos-parameter
If **queueCreateInfoCount** is not 0, **pQueueCreateInfos** **must** be a valid pointer to an array of **queueCreateInfoCount** valid [VkDeviceQueueCreateInfo](#) structures
- VUID-VkDeviceCreateInfo-ppEnabledLayerNames-parameter
If **enabledLayerCount** is not 0, **ppEnabledLayerNames** **must** be a valid pointer to an array of **enabledLayerCount** null-terminated UTF-8 strings
- VUID-VkDeviceCreateInfo-ppEnabledExtensionNames-parameter
If **enabledExtensionCount** is not 0, **ppEnabledExtensionNames** **must** be a valid pointer to an array of **enabledExtensionCount** null-terminated UTF-8 strings
- VUID-VkDeviceCreateInfo-pEnabledFeatures-parameter
If **pEnabledFeatures** is not **NULL**, **pEnabledFeatures** **must** be a valid pointer to a valid [VkPhysicalDeviceFeatures](#) structure

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkDeviceCreateFlags;
```

VkDeviceCreateFlags is a bitmask type for setting a mask, but is currently reserved for future use.

A logical device **can** be created that connects to one or more physical devices by adding a

`VkDeviceGroupDeviceCreateInfo` structure to the `pNext` chain of `VkDeviceCreateInfo`. The `VkDeviceGroupDeviceCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDeviceGroupDeviceCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           physicalDeviceCount;
    const VkPhysicalDevice* pPhysicalDevices;
} VkDeviceGroupDeviceCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `physicalDeviceCount` is the number of elements in the `pPhysicalDevices` array.
- `pPhysicalDevices` is a pointer to an array of physical device handles belonging to the same device group.

The elements of the `pPhysicalDevices` array are an ordered list of the physical devices that the logical device represents. These **must** be a subset of a single device group, and need not be in the same order as they were enumerated. The order of the physical devices in the `pPhysicalDevices` array determines the *device index* of each physical device, with element `i` being assigned a device index of `i`. Certain commands and structures refer to one or more physical devices by using device indices or *device masks* formed using device indices.

A logical device created without using `VkDeviceGroupDeviceCreateInfo`, or with `physicalDeviceCount` equal to zero, is equivalent to a `physicalDeviceCount` of one and `pPhysicalDevices` pointing to the `physicalDevice` parameter to `vkCreateDevice`. In particular, the device index of that physical device is zero.

Valid Usage

- VUID-VkDeviceGroupDeviceCreateInfo-pPhysicalDevices-00375
Each element of `pPhysicalDevices` **must** be unique
- VUID-VkDeviceGroupDeviceCreateInfo-pPhysicalDevices-00376
All elements of `pPhysicalDevices` **must** be in the same device group as enumerated by `vkEnumeratePhysicalDeviceGroups`
- VUID-VkDeviceGroupDeviceCreateInfo-physicalDeviceCount-00377
If `physicalDeviceCount` is not 0, the `physicalDevice` parameter of `vkCreateDevice` **must** be an element of `pPhysicalDevices`

Valid Usage (Implicit)

- VUID-VkDeviceGroupDeviceCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_GROUP_DEVICE_CREATE_INFO`

- VUID-VkDeviceGroupDeviceCreateInfo-pPhysicalDevices-parameter

If `physicalDeviceCount` is not 0, `pPhysicalDevices` **must** be a valid pointer to an array of `physicalDeviceCount` valid `VkPhysicalDevice` handles

Data structures for objects are reserved by the implementation at device creation time. The application **must** provide upper bounds on numbers of objects and other limits at device creation time. To reserve data structures for use by objects created from this device, add a `VkDeviceObjectReservationCreateInfo` structure to the `pNext` chain of the `VkDeviceCreateInfo` structure.

```

// Provided by VKSC_VERSION_1_0
typedef struct VkDeviceObjectReservationCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    uint32_t                  pipelineCacheCreateInfoCount;
    const VkPipelineCacheCreateInfo* pPipelineCacheCreateInfos;
    uint32_t                  pipelinePoolSizeCount;
    const VkPipelinePoolSize* pPipelinePoolSizes;
    uint32_t                  semaphoreRequestCount;
    uint32_t                  commandBufferRequestCount;
    uint32_t                  fenceRequestCount;
    uint32_t                  deviceMemoryRequestCount;
    uint32_t                  bufferRequestCount;
    uint32_t                  imageRequestCount;
    uint32_t                  eventRequestCount;
    uint32_t                  queryPoolRequestCount;
    uint32_t                  bufferViewRequestCount;
    uint32_t                  imageViewRequestCount;
    uint32_t                  layeredImageViewRequestCount;
    uint32_t                  pipelineCacheRequestCount;
    uint32_t                  pipelineLayoutRequestCount;
    uint32_t                  renderPassRequestCount;
    uint32_t                  graphicsPipelineRequestCount;
    uint32_t                  computePipelineRequestCount;
    uint32_t                  descriptorSetLayoutRequestCount;
    uint32_t                  samplerRequestCount;
    uint32_t                  descriptorPoolRequestCount;
    uint32_t                  descriptorSetRequestCount;
    uint32_t                  framebufferRequestCount;
    uint32_t                  commandPoolRequestCount;
    uint32_t                  samplerYcbcrConversionRequestCount;
    uint32_t                  surfaceRequestCount;
    uint32_t                  swapchainRequestCount;
    uint32_t                  displayModeRequestCount;
    uint32_t                  subpassDescriptionRequestCount;
    uint32_t                  attachmentDescriptionRequestCount;
    uint32_t                  descriptorSetLayoutBindingRequestCount;
    uint32_t                  descriptorSetLayoutBindingLimit;
    uint32_t                  maxImageViewMipLevels;
    uint32_t                  maxImageViewArrayLayers;
    uint32_t                  maxLayeredImageViewMipLevels;
    uint32_t                  maxOcclusionQueriesPerPool;
    uint32_t                  maxPipelineStatisticsQueriesPerPool;
    uint32_t                  maxTimestampQueriesPerPool;
    uint32_t                  maxImmutableSamplersPerDescriptorSetLayout;
} VkDeviceObjectReservationCreateInfo;

```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.

- `pipelineCacheCreateInfoCount` is the length of the `pPipelineCacheCreateInfos` array.
- `pPipelineCacheCreateInfos` is a pointer to an array of `VkPipelineCacheCreateInfo` structures that contain the creation information of the pipeline caches that **can** be created on this device.
- `pipelinePoolSizeCount` is the length of the `pPipelinePoolSizes` array.
- `pPipelinePoolSizes` is a pointer to an array of `VkPipelinePoolSize` structures requesting memory be reserved for pipelines of the specified sizes.
- `semaphoreRequestCount` is the requested maximum number of `VkSemaphore` objects that **can** exist at the same time.
- `commandBufferRequestCount` is the requested maximum number of `VkCommandBuffer` objects that **can** be reserved by all `VkCommandPool` objects.
- `fenceRequestCount` is the requested maximum number of `VkFence` objects that **can** exist at the same time.
- `deviceMemoryRequestCount` is the requested maximum number of `VkDeviceMemory` objects that **can** exist at the same time.
- `bufferRequestCount` is the requested maximum number of `VkBuffer` objects that **can** exist at the same time.
- `imageRequestCount` is the requested maximum number of `VkImage` objects that **can** exist at the same time.
- `eventRequestCount` is the requested maximum number of `VkEvent` objects that **can** exist at the same time.
- `queryPoolRequestCount` is the requested maximum number of `VkQueryPool` objects that **can** exist at the same time.
- `bufferViewRequestCount` is the requested maximum number of `VkBufferView` objects that **can** exist at the same time.
- `imageViewRequestCount` is the requested maximum number of `VkImageView` objects that **can** exist at the same time.
- `layeredImageViewRequestCount` is the requested maximum number `VkImageView` objects created with `VkImageViewCreateInfo::subresourceRange.layerCount` greater than 1 that **can** exist at the same time.
- `pipelineCacheRequestCount` is the requested maximum number of `VkPipelineCache` objects that **can** exist at the same time.
- `pipelineLayoutRequestCount` is the requested maximum number of `VkPipelineLayout` objects that **can** exist at the same time.
- `renderPassRequestCount` is the requested maximum number of `VkRenderPass` objects that **can** exist at the same time.
- `graphicsPipelineRequestCount` is the requested maximum number of graphics `VkPipeline` objects that **can** exist at the same time.
- `computePipelineRequestCount` is the requested maximum number of compute `VkPipeline` objects that **can** exist at the same time.
- `descriptorSetLayoutRequestCount` is the requested maximum number of `VkDescriptorSetLayout`

objects that **can** exist at the same time.

- `samplerRequestCount` is the requested maximum number of `VkSampler` objects that **can** exist at the same time.
- `descriptorPoolRequestCount` is the requested maximum number of `VkDescriptorPool` objects that **can** exist at the same time.
- `descriptorSetRequestCount` is the requested maximum number of `VkDescriptorSet` objects that **can** exist at the same time.
- `framebufferRequestCount` is the requested maximum number of `VkFramebuffer` objects that **can** exist at the same time.
- `commandPoolRequestCount` is the requested maximum number of `VkCommandPool` objects that **can** exist at the same time.
- `samplerYcbcrConversionRequestCount` is the requested maximum number of `VkSamplerYcbcrConversion` objects that **can** exist at the same time.
- `surfaceRequestCount` is legacy and implementations **must** ignore it.
- `swapchainRequestCount` is the requested maximum number of `VkSwapchainKHR` objects that **can** exist at the same time.
- `displayModeRequestCount` is legacy and implementations **must** ignore it.
- `subpassDescriptionRequestCount` is the requested maximum sum of all `VkRenderPassCreateInfo2::subpassCount` values across all `VkRenderPass` objects that **can** exist at the same time.
- `attachmentDescriptionRequestCount` is the requested maximum sum of all `VkRenderPassCreateInfo2::attachmentCount` values across all `VkRenderPass` objects that **can** exist at the same time.
- `descriptorSetLayoutBindingRequestCount` is the requested maximum sum of all `VkDescriptorSetLayoutCreateInfo::bindingCount` values across all `VkDescriptorSetLayout` objects that **can** exist at the same time.
- `descriptorSetLayoutBindingLimit` is one greater than the maximum value of `VkDescriptorSetLayoutBinding::binding` that **can** be used.
- `maxImageViewMipLevels` is the maximum value of `VkImageViewCreateInfo::subresourceRange.levelCount` that **can** be used.
- `maxImageViewArrayLayers` is the maximum value of `VkImageViewCreateInfo::subresourceRange.layerCount` that **can** be used.
- `maxLayeredImageViewMipLevels` is the maximum value of `VkImageViewCreateInfo::subresourceRange.levelCount` that **can** be used when `VkImageViewCreateInfo::subresourceRange.layerCount` is greater than 1.
- `maxOcclusionQueriesPerPool` is the requested maximum number of `VK_QUERY_TYPE_OCCLUSION` queries that **can** exist at the same time in a single query pool.
- `maxPipelineStatisticsQueriesPerPool` is the requested maximum number of `VK_QUERY_TYPE_PIPELINE_STATISTICS` queries that **can** exist at the same time in a single query pool.

- `maxTimestampQueriesPerPool` is the requested maximum number of `VK_QUERY_TYPE_TIMESTAMP` queries that **can** exist at the same time in a single query pool.
- `maxImmutableSamplersPerDescriptorSetLayout` is the requested maximum number of immutable samplers that can be used across all bindings in a descriptor set layout.

Multiple `VkDeviceObjectReservationCreateInfo` structures **can** be chained together. The maximum value from all instances of `maxImageViewMipLevels`, `maxImageViewArrayLayers`, `maxLayeredImageViewMipLevels`, `descriptorSetLayoutBindingLimit`, `maxOcclusionQueriesPerPool`, `maxPipelineStatisticsQueriesPerPool`, `maxTimestampQueriesPerPool`, and `maxImmutableSamplersPerDescriptorSetLayout` will be reserved. For the remaining members, the sum of the requested resources from all instances of `VkDeviceObjectReservationCreateInfo` will be reserved.

If `VkPhysicalDeviceVulkanSC10Properties::deviceDestroyFreesMemory` is `VK_TRUE`, the reserved memory is returned to the system when the device is destroyed, otherwise it **may** not be returned to the system until the process is terminated.

Valid Usage

- VUID-VkDeviceObjectReservationCreateInfo-maxImageViewArrayLayers-05014 `maxImageViewArrayLayers` **must** be less than or equal to `VkPhysicalDeviceLimits::maxImageArrayLayers`
- VUID-VkDeviceObjectReservationCreateInfo-maxImageViewMipLevels-05015 `maxImageViewMipLevels` **must** be less than or equal to the number of levels in the complete mipmap chain based on the maximum of `VkPhysicalDeviceLimits::maxImageDimension1D`, `maxImageDimension2D`, `maxImageDimension3D`, and `maxImageDimensionCube`
- VUID-VkDeviceObjectReservationCreateInfo-maxLayeredImageViewMipLevels-05016 `maxLayeredImageViewMipLevels` **must** be less than or equal to the number of levels in the complete mipmap chain based on `VkPhysicalDeviceLimits::maxImageDimension1D`, `maxImageDimension2D`, `maxImageDimension3D`, and `maxImageDimensionCube`
- VUID-VkDeviceObjectReservationCreateInfo-subpassDescriptionRequestCount-05017 `subpassDescriptionRequestCount` **must** be less than or equal to `renderPassRequestCount` multiplied by `VkPhysicalDeviceVulkanSC10Properties::maxRenderPassSubpasses`
- VUID-VkDeviceObjectReservationCreateInfo-attachmentDescriptionRequestCount-05018 `attachmentDescriptionRequestCount` **must** be less than or equal to `renderPassRequestCount` multiplied by `VkPhysicalDeviceVulkanSC10Properties::maxFramebufferAttachments`

Valid Usage (Implicit)

- VUID-VkDeviceObjectReservationCreateInfo-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_OBJECT_RESERVATION_CREATE_INFO`
- VUID-VkDeviceObjectReservationCreateInfo-pPipelineCacheCreateInfos-parameter If `pipelineCacheCreateInfoCount` is not 0, `pPipelineCacheCreateInfos` **must** be a valid pointer to an array of `pipelineCacheCreateInfoCount` valid `VkPipelineCacheCreateInfo`

structures

- VUID-VkDeviceObjectReservationCreateInfo-pPipelinePoolSizes-parameter

If `pipelinePoolSizeCount` is not 0, `pPipelinePoolSizes` **must** be a valid pointer to an array of `pipelinePoolSizeCount` valid `VkPipelinePoolSize` structures

If the `pNext` chain of `VkDeviceObjectReservationCreateInfo` includes a `VkPerformanceQueryReservationInfoKHR` structure, then the structure indicates upper bounds on the number of performance queries that **can** exist at the same time in a query pool.

The `VkPerformanceQueryReservationInfoKHR` structure is defined as:

```
// Provided by VKSC_VERSION_1_0 with VK_KHR_performance_query
typedef struct VkPerformanceQueryReservationInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           maxPerformanceQueriesPerPool;
} VkPerformanceQueryReservationInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxPerformanceQueriesPerPool` is the requested maximum number of `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR` queries that **can** exist at the same time in a single query pool.

If the `VkDeviceObjectReservationCreateInfo::pNext` chain does not include this structure, then `maxPerformanceQueriesPerPool` defaults to 0.

Multiple `VkPerformanceQueryReservationInfoKHR` structures can be chained together. The maximum value from all instances of `maxPerformanceQueriesPerPool` will be reserved.

Valid Usage (Implicit)

- VUID-VkPerformanceQueryReservationInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PERFORMANCE_QUERY_RESERVATION_INFO_KHR`

If the `pNext` chain of `VkDeviceObjectReservationCreateInfo` includes a `VkDeviceSemaphoreSciSyncPoolReservationCreateInfoNV` structure, then the structure indicates the maximum number of `VkSemaphoreSciSyncPoolNV` objects that **can** exist at the same time.

The `VkDeviceSemaphoreSciSyncPoolReservationCreateInfoNV` structure is defined as:


```
// Provided by VKSC_VERSION_1_0 with VK_NV_external_sci_sync2
typedef struct VkDeviceSemaphoreSciSyncPoolReservationCreateInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           semaphoreSciSyncPoolRequestCount;
} VkDeviceSemaphoreSciSyncPoolReservationCreateInfoNV;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `semaphoreSciSyncPoolRequestCount` is the requested maximum number of `VkSemaphoreSciSyncPoolNV` objects that **can** exist at the same time.

If the `VkDeviceObjectReservationCreateInfo::pNext` chain does not include this structure, then `semaphoreSciSyncPoolRequestCount` defaults to 0.

Multiple `VkDeviceSemaphoreSciSyncPoolReservationCreateInfoNV` structures **can** be chained together. The sum of the `semaphoreSciSyncPoolRequestCount` values from all instances of `VkDeviceSemaphoreSciSyncPoolReservationCreateInfoNV` will be reserved.

Valid Usage (Implicit)

- VUID-VkDeviceSemaphoreSciSyncPoolReservationCreateInfoNV-sType-sType **must** be `VK_STRUCTURE_TYPE_DEVICE_SEMAPHORE_SCI_SYNC_POOL_RESERVATION_CREATE_INFO_NV`

Memory for pipelines is reserved by the implementation at device creation time. The application specifies sizes to be reserved and a count for each size, and when a pipeline is created the application specifies which size to use.

```
// Provided by VKSC_VERSION_1_0
typedef struct VkPipelinePoolSize {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceSize        poolEntrySize;
    uint32_t           poolEntryCount;
} VkPipelinePoolSize;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `poolEntrySize` is the size to reserve for each entry.
- `poolEntryCount` is the number of entries to reserve.

Valid Usage (Implicit)

- VUID-VkPipelinePoolSize-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_POOL_SIZE`
- VUID-VkPipelinePoolSize-pNext-pNext
`pNext` **must** be `NULL`

5.2.2. Device Use

The following is a high-level list of `VkDevice` uses along with references on where to find more information:

- Creation of queues. See the [Queues](#) section below for further details.
- Creation and tracking of various synchronization constructs. See [Synchronization and Cache Control](#) for further details.
- Allocating, freeing, and managing memory. See [Memory Allocation](#) and [Resource Creation](#) for further details.
- Creation and destruction of command buffers and command buffer pools. See [Command Buffers](#) for further details.
- Creation, destruction, and management of graphics state. See [Pipelines](#) and [Resource Descriptors](#), among others, for further details.

5.2.3. Lost Device

A logical device **may** become *lost* for a number of implementation-specific reasons, indicating that pending and future command execution **may** fail and cause resources and backing memory to become undefined.

NOTE

[Fault Handling](#) can be used by the implementation to provide more information on the cause of a device becoming *lost*. Allowing applications to take appropriate corrective behavior for the cause of the device lost.

NOTE

Typical reasons for device loss will include things like execution timing out (to prevent denial of service), power management events, platform resource management, implementation errors.

Applications not adhering to [valid usage](#) may also result in device loss being reported, however this is not guaranteed. Even if device loss is reported, the system may be in an unrecoverable state, and further usage of the API is still considered invalid.

When this happens, certain commands will return `VK_ERROR_DEVICE_LOST`. After any such event, the logical device is considered *lost*. It is not possible to reset the logical device to a non-lost state, however the lost state is specific to a logical device (`VkDevice`), and the corresponding physical device (`VkPhysicalDevice`) **may** be otherwise unaffected.

In some cases, the physical device **may** also be lost, and attempting to create a new logical device will fail, returning `VK_ERROR_DEVICE_LOST`. This is usually indicative of a problem with the underlying implementation, or its connection to the host. If the physical device has not been lost, and a new logical device is successfully created from that physical device, it **must** be in the non-lost state.

NOTE

Whilst logical device loss **may** be recoverable, in the case of physical device loss, it is unlikely that an application will be able to recover unless additional, unaffected physical devices exist on the system. The error is largely informational and intended only to inform the application that a platform issue has occurred, and **should** be investigated further. For example, underlying hardware **may** have developed a fault or become physically disconnected from the rest of the system. In many cases, physical device loss **may** cause other more serious issues such as the operating system crashing; in which case it **may** not be reported via the Vulkan API.

When a device is lost, its child objects are not implicitly destroyed and their handles are still valid. Those objects **must** still be destroyed before their parents or the device **can** be destroyed (see the [Object Lifetime](#) section). The host address space corresponding to device memory mapped using `vkMapMemory` is still valid, and host memory accesses to these mapped regions are still valid, but the contents are undefined. It is still legal to call any API command on the device and child objects.

Once a device is lost, command execution **may** fail, and certain commands that return a `VkResult` **may** return `VK_ERROR_DEVICE_LOST`. These commands can be identified by the inclusion of `VK_ERROR_DEVICE_LOST` in the Return Codes section for each command. Commands that do not allow runtime errors **must** still operate correctly for valid usage and, if applicable, return valid data.

Commands that wait indefinitely for device execution (namely `vkDeviceWaitIdle`, `vkQueueWaitIdle`, `vkWaitForFences` or `vkAcquireNextImageKHR` with a maximum `timeout`, and `vkGetQueryPoolResults` with the `VK_QUERY_RESULT_WAIT_BIT` bit set in `flags`) **must** return in finite time even in the case of a lost device, and return either `VK_SUCCESS` or `VK_ERROR_DEVICE_LOST`. For any command that **may** return `VK_ERROR_DEVICE_LOST`, for the purpose of determining whether a command buffer is in the [pending state](#), or whether resources are considered in-use by the device, a return value of `VK_ERROR_DEVICE_LOST` is equivalent to `VK_SUCCESS`.

The content of any external memory objects that have been exported from or imported to a lost device become undefined. Objects on other logical devices or in other APIs which are associated with the same underlying memory resource as the external memory objects on the lost device are unaffected other than their content becoming undefined. The layout of subresources of images on other logical devices that are bound to `VkDeviceMemory` objects associated with the same underlying memory resources as external memory objects on the lost device becomes `VK_IMAGE_LAYOUT_UNDEFINED`.

The state of `VkSemaphore` objects on other logical devices created by [importing a semaphore payload](#) with temporary permanence which was exported from the lost device is undefined. The state of `VkSemaphore` objects on other logical devices that permanently share a semaphore payload with a `VkSemaphore` object on the lost device is undefined, and remains undefined following any subsequent signal operations. Implementations **must** ensure pending and subsequently submitted wait operations on such semaphores behave as defined in [Semaphore State Requirements For Wait Operations](#) for external semaphores not in a valid state for a wait operation.

5.2.4. Device Destruction

To destroy a device, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyDevice(
    VkDevice          device,
    const VkAllocationCallbacks* pAllocator);
```

- **device** is the logical device to destroy.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.

To ensure that no work is active on the device, **vkDeviceWaitIdle** can be used to gate the destruction of the device. Prior to destroying a device, an application is responsible for destroying/freeing any Vulkan objects with explicit **vkDestroy*** or **vkFree*** commands that were created using that device as the first parameter of the corresponding **vkCreate*** or **vkAllocate*** command.

NOTE

The lifetime of each of these objects is bound by the lifetime of the **VkDevice** object. Therefore, to avoid resource leaks, it is critical that an application explicitly free all of these resources prior to calling **vkDestroyDevice**.

Valid Usage

- VUID-vkDestroyDevice-device-05137
All child objects created on **device** that can be destroyed or freed **must** have been destroyed or freed prior to destroying **device**

Valid Usage (Implicit)

- VUID-vkDestroyDevice-device-parameter
If **device** is not **NULL**, **device** **must** be a valid **VkDevice** handle
- VUID-vkDestroyDevice-pAllocator-null
pAllocator **must** be **NULL**

Host Synchronization

- Host access to **device** **must** be externally synchronized
- Host access to all **VkQueue** objects created from **device** **must** be externally synchronized

5.3. Queues

5.3.1. Queue Family Properties

As discussed in the [Physical Device Enumeration](#) section above, the `vkGetPhysicalDeviceQueueFamilyProperties` command is used to retrieve details about the queue families and queues supported by a device.

Each index in the `pQueueFamilyProperties` array returned by `vkGetPhysicalDeviceQueueFamilyProperties` describes a unique queue family on that physical device. These indices are used when creating queues, and they correspond directly with the `queueFamilyIndex` that is passed to the `vkCreateDevice` command via the `VkDeviceQueueCreateInfo` structure as described in the [Queue Creation](#) section below.

Grouping of queue families within a physical device is implementation-dependent.

NOTE

The general expectation is that a physical device groups all queues of matching capabilities into a single family. However, while implementations **should** do this, it is possible that a physical device **may** return two separate queue families with the same capabilities.

Once an application has identified a physical device with the queue(s) that it desires to use, it will create those queues in conjunction with a logical device. This is described in the following section.

5.3.2. Queue Creation

Creating a logical device also creates the queues associated with that device. The queues to create are described by a set of `VkDeviceQueueCreateInfo` structures that are passed to `vkCreateDevice` in `pQueueCreateInfos`. Queues **cannot** be independently destroyed, and are instead destroyed with the `VkDevice` that they were created from.

Queues are represented by `VkQueue` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_HANDLE(VkQueue)
```

The `VkDeviceQueueCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDeviceQueueCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceQueueCreateFlags flags;
    uint32_t           queueFamilyIndex;
    uint32_t           queueCount;
    const float*        pQueuePriorities;
} VkDeviceQueueCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask indicating behavior of the queues.
- `queueFamilyIndex` is an unsigned integer indicating the index of the queue family in which to create the queues on this device. This index corresponds to the index of an element of the `pQueueFamilyProperties` array that was returned by `vkGetPhysicalDeviceQueueFamilyProperties`.
- `queueCount` is an unsigned integer specifying the number of queues to create in the queue family indicated by `queueFamilyIndex`, and with the behavior specified by `flags`.
- `pQueuePriorities` is a pointer to an array of `queueCount` normalized floating-point values, specifying priorities of work that will be submitted to each created queue. See [Queue Priority](#) for more information.

Valid Usage

- VUID-VkDeviceQueueCreateInfo-queueFamilyIndex-00381
`queueFamilyIndex` **must** be less than `pQueueFamilyPropertyCount` returned by `vkGetPhysicalDeviceQueueFamilyProperties`
- VUID-VkDeviceQueueCreateInfo-queueCount-00382
`queueCount` **must** be less than or equal to the `queueCount` member of the `VkQueueFamilyProperties` structure, as returned by `vkGetPhysicalDeviceQueueFamilyProperties` in the `pQueueFamilyProperties[queueFamilyIndex]`
- VUID-VkDeviceQueueCreateInfo-pQueuePriorities-00383
Each element of `pQueuePriorities` **must** be between `0.0` and `1.0` inclusive
- VUID-VkDeviceQueueCreateInfo-flags-02861
If the `protectedMemory` feature is not enabled, the `VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT` bit of `flags` **must** not be set
- VUID-VkDeviceQueueCreateInfo-flags-06449
If `flags` includes `VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT`, `queueFamilyIndex` **must** be the index of a queue family that includes the `VK_QUEUE_PROTECTED_BIT` capability

Valid Usage (Implicit)

- VUID-VkDeviceQueueCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO`
- VUID-VkDeviceQueueCreateInfo-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkDeviceQueueGlobalPriorityCreateInfo`
- VUID-VkDeviceQueueCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkDeviceQueueCreateInfo-flags-parameter
`flags` **must** be a valid combination of `VkDeviceQueueCreateFlagBits` values

- VUID-VkDeviceQueueCreateInfo-pQueuePriorities-parameter
`pQueuePriorities` **must** be a valid pointer to an array of `queueCount` `float` values
- VUID-VkDeviceQueueCreateInfo-queueCount-arraylength
`queueCount` **must** be greater than 0

Bits which **can** be set in `VkDeviceQueueCreateInfo::flags`, specifying usage behavior of a queue, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkDeviceQueueCreateFlagBits {
    // Provided by VK_VERSION_1_1
    VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT = 0x00000001,
} VkDeviceQueueCreateFlagBits;
```

- `VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT` specifies that the device queue is a protected-capable queue.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkDeviceQueueCreateFlags;
```

`VkDeviceQueueCreateFlags` is a bitmask type for setting a mask of zero or more `VkDeviceQueueCreateFlagBits`.

Queues **can** be created with a system-wide priority by adding a `VkDeviceQueueGlobalPriorityCreateInfo` structure to the `pNext` chain of `VkDeviceQueueCreateInfo`.

The `VkDeviceQueueGlobalPriorityCreateInfo` structure is defined as:

```
typedef struct VkDeviceQueueGlobalPriorityCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkQueueGlobalPriority globalPriority;
} VkDeviceQueueGlobalPriorityCreateInfo;
```

```
// Provided by VK_KHR_global_priority
typedef VkDeviceQueueGlobalPriorityCreateInfo
VkDeviceQueueGlobalPriorityCreateInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `globalPriority` is the system-wide priority associated to these queues as specified by `VkQueueGlobalPriority`

Queues created without specifying `VkDeviceQueueGlobalPriorityCreateInfo` will default to

VK_QUEUE_GLOBAL_PRIORITY_MEDIUM.

Valid Usage (Implicit)

- VUID-VkDeviceQueueGlobalPriorityCreateInfo-sType-sType
sType **must** be VK_STRUCTURE_TYPE_DEVICE_QUEUE_GLOBAL_PRIORITY_CREATE_INFO
- VUID-VkDeviceQueueGlobalPriorityCreateInfo-globalPriority-parameter
globalPriority **must** be a valid VkQueueGlobalPriority value

Possible values of `VkDeviceQueueGlobalPriorityCreateInfo::globalPriority`, specifying a system-wide priority level are:

```
typedef enum VkQueueGlobalPriority {  
    VK_QUEUE_GLOBAL_PRIORITY_LOW = 128,  
    VK_QUEUE_GLOBAL_PRIORITY_MEDIUM = 256,  
    VK_QUEUE_GLOBAL_PRIORITY_HIGH = 512,  
    VK_QUEUE_GLOBAL_PRIORITY_REALTIME = 1024,  
    // Provided by VK_KHR_global_priority  
    VK_QUEUE_GLOBAL_PRIORITY_LOW_KHR = VK_QUEUE_GLOBAL_PRIORITY_LOW,  
    // Provided by VK_KHR_global_priority  
    VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_KHR = VK_QUEUE_GLOBAL_PRIORITY_MEDIUM,  
    // Provided by VK_KHR_global_priority  
    VK_QUEUE_GLOBAL_PRIORITY_HIGH_KHR = VK_QUEUE_GLOBAL_PRIORITY_HIGH,  
    // Provided by VK_KHR_global_priority  
    VK_QUEUE_GLOBAL_PRIORITY_REALTIME_KHR = VK_QUEUE_GLOBAL_PRIORITY_REALTIME,  
} VkQueueGlobalPriority;
```

```
// Provided by VK_KHR_global_priority  
typedef VkQueueGlobalPriority VkQueueGlobalPriorityKHR;
```

Priority values are sorted in ascending order. A comparison operation on the enum values can be used to determine the priority order.

- VK_QUEUE_GLOBAL_PRIORITY_LOW is below the system default. Useful for non-interactive tasks.
- VK_QUEUE_GLOBAL_PRIORITY_MEDIUM is the system default priority.
- VK_QUEUE_GLOBAL_PRIORITY_HIGH is above the system default.
- VK_QUEUE_GLOBAL_PRIORITY_REALTIME is the highest priority. Useful for critical tasks.

Queues with higher system priority **may** be allotted more processing time than queues with lower priority. An implementation **may** allow a higher-priority queue to starve a lower-priority queue until the higher-priority queue has no further commands to execute.

Priorities imply no ordering or scheduling constraints.

No specific guarantees are made about higher priority queues receiving more processing time or

better quality of service than lower priority queues.

The global priority level of a queue takes precedence over the per-process queue priority ([VkDeviceQueueCreateInfo::pQueuePriorities](#)).

Abuse of this feature **may** result in starving the rest of the system of implementation resources. Therefore, the driver implementation **may** deny requests to acquire a priority above the default priority ([VK_QUEUE_GLOBAL_PRIORITY_MEDIUM](#)) if the caller does not have sufficient privileges. In this scenario [VK_ERROR_NOT_PERMITTED](#) is returned.

The driver implementation **may** fail the queue allocation request if resources required to complete the operation have been exhausted (either by the same process or a different process). In this scenario [VK_ERROR_INITIALIZATION_FAILED](#) is returned.

If the [globalPriorityQuery](#) feature is enabled and the requested global priority is not reported via [VkQueueFamilyGlobalPriorityProperties](#), the driver implementation **must** fail the queue creation. In this scenario, [VK_ERROR_INITIALIZATION_FAILED](#) is returned.

To retrieve a handle to a [VkQueue](#) object, call:

```
// Provided by VK_VERSION_1_0
void vkGetDeviceQueue(
    VkDevice          device,
    uint32_t          queueFamilyIndex,
    uint32_t          queueIndex,
    VkQueue*          pQueue);
```

- [device](#) is the logical device that owns the queue.
- [queueFamilyIndex](#) is the index of the queue family to which the queue belongs.
- [queueIndex](#) is the index within this queue family of the queue to retrieve.
- [pQueue](#) is a pointer to a [VkQueue](#) object that will be filled with the handle for the requested queue.

[vkGetDeviceQueue](#) **must** only be used to get queues that were created with the [flags](#) parameter of [VkDeviceQueueCreateInfo](#) set to zero. To get queues that were created with a non-zero [flags](#) parameter use [vkGetDeviceQueue2](#).

Valid Usage

- VUID-vkGetDeviceQueue-queueFamilyIndex-00384
[queueFamilyIndex](#) **must** be one of the queue family indices specified when [device](#) was created, via the [VkDeviceQueueCreateInfo](#) structure
- VUID-vkGetDeviceQueue-queueIndex-00385
[queueIndex](#) **must** be less than the value of [VkDeviceQueueCreateInfo::queueCount](#) for the queue family indicated by [queueFamilyIndex](#) when [device](#) was created
- VUID-vkGetDeviceQueue-flags-01841

`VkDeviceQueueCreateInfo::flags` **must** have been zero when `device` was created

Valid Usage (Implicit)

- VUID-vkGetDeviceQueue-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetDeviceQueue-pQueue-parameter
`pQueue` **must** be a valid pointer to a `VkQueue` handle

To retrieve a handle to a `VkQueue` object with specific `VkDeviceQueueCreateFlags` creation flags, call:

```
// Provided by VK_VERSION_1_1
void vkGetDeviceQueue2(
    VkDevice                device,
    const VkDeviceQueueInfo2* pQueueInfo,
    VkQueue*                pQueue);
```

- `device` is the logical device that owns the queue.
- `pQueueInfo` is a pointer to a `VkDeviceQueueInfo2` structure, describing parameters of the device queue to be retrieved.
- `pQueue` is a pointer to a `VkQueue` object that will be filled with the handle for the requested queue.

Valid Usage (Implicit)

- VUID-vkGetDeviceQueue2-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetDeviceQueue2-pQueueInfo-parameter
`pQueueInfo` **must** be a valid pointer to a valid `VkDeviceQueueInfo2` structure
- VUID-vkGetDeviceQueue2-pQueue-parameter
`pQueue` **must** be a valid pointer to a `VkQueue` handle

The `VkDeviceQueueInfo2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDeviceQueueInfo2 {
    VkStructureType      sType;
    const void*          pNext;
    VkDeviceQueueCreateFlags flags;
    uint32_t             queueFamilyIndex;
    uint32_t             queueIndex;
} VkDeviceQueueInfo2;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is `NULL` or a pointer to a structure extending this structure. The **pNext** chain of [VkDeviceQueueInfo2](#) **can** be used to provide additional device queue parameters to [vkGetDeviceQueue2](#).
- **flags** is a [VkDeviceQueueCreateFlags](#) value indicating the flags used to create the device queue.
- **queueFamilyIndex** is the index of the queue family to which the queue belongs.
- **queueIndex** is the index of the queue to retrieve from within the set of queues that share both the queue family and flags specified.

The queue returned by [vkGetDeviceQueue2](#) **must** have the same **flags** value from this structure as that used at device creation time in a [VkDeviceQueueCreateInfo](#) structure.

NOTE

Normally, if you create both protected-capable and non-protected-capable queues with the same family, they are treated as separate lists of queues and **queueIndex** is relative to the start of the list of queues specified by both **queueFamilyIndex** and **flags**. However, for historical reasons, some implementations may exhibit different behavior. These divergent implementations instead concatenate the lists of queues and treat **queueIndex** as relative to the start of the first list of queues with the given **queueFamilyIndex**. This only matters in cases where an application has created both protected-capable and non-protected-capable queues from the same queue family.

For such divergent implementations, the maximum value of **queueIndex** is equal to the sum of [VkDeviceQueueCreateInfo::queueCount](#) minus one, for all [VkDeviceQueueCreateInfo](#) structures that share a common **queueFamilyIndex**.

Such implementations will return `NULL` for either the protected or unprotected queues when calling [vkGetDeviceQueue2](#) with **queueIndex** in the range zero to [VkDeviceQueueCreateInfo::queueCount](#) minus one. In cases where these implementations returned `NULL`, the corresponding queues are instead located in the extended range described in the preceding two paragraphs.

This behavior will not be observed on any driver that has passed Vulkan conformance test suite version 1.3.3.0, or any subsequent version. This information can be found by querying [VkPhysicalDeviceDriverProperties::conformanceVersion](#).

Valid Usage

- VUID-VkDeviceQueueInfo2-queueFamilyIndex-01842
`queueFamilyIndex` **must** be one of the queue family indices specified when `device` was created, via the `VkDeviceQueueCreateInfo` structure
- VUID-VkDeviceQueueInfo2-flags-06225
`flags` **must** be equal to `VkDeviceQueueCreateInfo::flags` for a `VkDeviceQueueCreateInfo` structure for the queue family indicated by `queueFamilyIndex` when `device` was created
- VUID-VkDeviceQueueInfo2-queueIndex-01843
`queueIndex` **must** be less than `VkDeviceQueueCreateInfo::queueCount` for the corresponding queue family and flags indicated by `queueFamilyIndex` and `flags` when `device` was created

Valid Usage (Implicit)

- VUID-VkDeviceQueueInfo2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_QUEUE_INFO_2`
- VUID-VkDeviceQueueInfo2-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkDeviceQueueInfo2-flags-parameter
`flags` **must** be a valid combination of `VkDeviceQueueCreateFlagBits` values

5.3.3. Queue Family Index

The queue family index is used in multiple places in Vulkan in order to tie operations to a specific family of queues.

When retrieving a handle to the queue via `vkGetDeviceQueue`, the queue family index is used to select which queue family to retrieve the `VkQueue` handle from as described in the previous section.

When creating a `VkCommandPool` object (see [Command Pools](#)), a queue family index is specified in the `VkCommandPoolCreateInfo` structure. Command buffers from this pool **can** only be submitted on queues corresponding to this queue family.

When creating `VkImage` (see [Images](#)) and `VkBuffer` (see [Buffers](#)) resources, a set of queue families is included in the `VkImageCreateInfo` and `VkBufferCreateInfo` structures to specify the queue families that **can** access the resource.

When inserting a `VkBufferMemoryBarrier` or `VkImageMemoryBarrier` (see [Pipeline Barriers](#)), a source and destination queue family index is specified to allow the ownership of a buffer or image to be transferred from one queue family to another. See the [Resource Sharing](#) section for details.

5.3.4. Queue Priority

Each queue is assigned a priority, as set in the `VkDeviceQueueCreateInfo` structures when creating the device. The priority of each queue is a normalized floating-point value between 0.0 and 1.0,

which is then translated to a discrete priority level by the implementation. Higher values indicate a higher priority, with 0.0 being the lowest priority and 1.0 being the highest.

Within the same device, queues with higher priority **may** be allotted more processing time than queues with lower priority. The implementation makes no guarantees with regards to ordering or scheduling among queues with the same priority, other than the constraints defined by any [explicit synchronization primitives](#). The implementation makes no guarantees with regards to queues across different devices.

An implementation **may** allow a higher-priority queue to starve a lower-priority queue on the same **VkDevice** until the higher-priority queue has no further commands to execute. The relationship of queue priorities **must** not cause queues on one **VkDevice** to starve queues on another **VkDevice**.

No specific guarantees are made about higher priority queues receiving more processing time or better quality of service than lower priority queues.

5.3.5. Queue Submission

Work is submitted to a queue via *queue submission* commands such as [vkQueueSubmit2KHR](#) or [vkQueuePresentKHR](#) or [vkQueueSubmit](#). Queue submission commands define a set of *queue operations* to be executed by the underlying physical device, including synchronization with semaphores and fences.

Submission commands take as parameters a target queue, zero or more *batches* of work, and an **optional** fence to signal upon completion. Each batch consists of three distinct parts:

1. Zero or more semaphores to wait on before execution of the rest of the batch.
 - If present, these describe a [semaphore wait operation](#).
2. Zero or more work items to execute.
 - If present, these describe a *queue operation* matching the work described.
3. Zero or more semaphores to signal upon completion of the work items.
 - If present, these describe a [semaphore signal operation](#).

If a fence is present in a queue submission, it describes a [fence signal operation](#).

All work described by a queue submission command **must** be submitted to the queue before the command returns.

5.3.6. Queue Destruction

Queues are created along with a logical device during **vkCreateDevice**. All queues associated with a logical device are destroyed when **vkDestroyDevice** is called on that device.

Chapter 6. Command Buffers

Command buffers are objects used to record commands which **can** be subsequently submitted to a device queue for execution. There are two levels of command buffers - *primary command buffers*, which **can** execute secondary command buffers, and which are submitted to queues, and *secondary command buffers*, which **can** be executed by primary command buffers, and which are not directly submitted to queues.

Command buffers are represented by `VkCommandBuffer` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_HANDLE(VkCommandBuffer)
```

Recorded commands include commands to bind pipelines and descriptor sets to the command buffer, commands to modify dynamic state, commands to draw (for graphics rendering), commands to dispatch (for compute), commands to execute secondary command buffers (for primary command buffers only), commands to copy buffers and images, and other commands.

Each command buffer manages state independently of other command buffers. There is no inheritance of state across primary and secondary command buffers, or between secondary command buffers. When a command buffer begins recording, all state in that command buffer is undefined. When secondary command buffer(s) are recorded to execute on a primary command buffer, the secondary command buffer inherits no state from the primary command buffer, and all state of the primary command buffer is undefined after an execute secondary command buffer command is recorded. There is one exception to this rule - if the primary command buffer is inside a render pass instance, then the render pass and subpass state is not disturbed by executing secondary command buffers. For state dependent commands (such as draws and dispatches), any state consumed by those commands **must** not be undefined.

Unless otherwise specified, and without explicit synchronization, the various commands submitted to a queue via command buffers **may** execute in arbitrary order relative to each other, and/or concurrently. Also, the memory side effects of those commands **may** not be directly visible to other commands without explicit memory dependencies. This is true within a command buffer, and across command buffers submitted to a given queue. See [the synchronization chapter](#) for information on [implicit](#) and explicit synchronization between commands.

6.1. Command Buffer Lifecycle

Each command buffer is always in one of the following states:

Initial

When a command buffer is [allocated](#), it is in the *initial state*. Some commands are able to *reset* a command buffer (or a set of command buffers) back to this state from any of the executable, recording or invalid state. Command buffers in the initial state **can** only be moved to the recording state, or freed.

Recording

[vkBeginCommandBuffer](#) changes the state of a command buffer from the initial state to the *recording state*. Once a command buffer is in the recording state, [vkCmd*](#) commands **can** be used to record to the command buffer.

Executable

[vkEndCommandBuffer](#) ends the recording of a command buffer, and moves it from the recording state to the *executable state*. Executable command buffers **can** be [submitted](#), reset, or [recorded to another command buffer](#).

Pending

[Queue submission](#) of a command buffer changes the state of a command buffer from the executable state to the *pending state*. Whilst in the pending state, applications **must** not attempt to modify the command buffer in any way - as the device **may** be processing the commands recorded to it. Once execution of a command buffer completes, the command buffer either reverts back to the *executable state*, or if it was recorded with [VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT](#), it moves to the *invalid state*. A [synchronization command](#) **should** be used to detect when this occurs.

Invalid

Other than [VkQueryPool](#), [VkDescriptorPool](#), [VkDeviceMemory](#), or [VkCommandPool](#) objects, destroying or freeing any object or memory that **may** be accessed when the command buffer is accessed (e.g. an object bound to the command buffer) will transition the state of that command buffer to the *invalid state*. The command buffer **must** not be in the *pending state* when this happens. Command buffers in the invalid state **can** only be reset or freed.

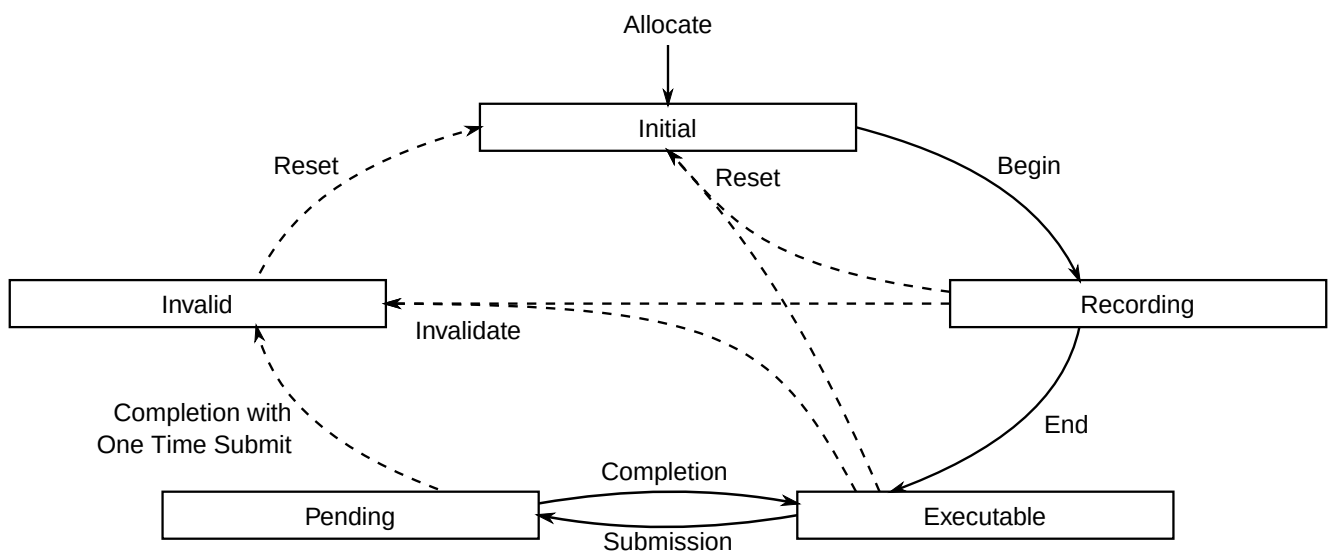


Figure 1. Lifecycle of a command buffer

Any given command that operates on a command buffer has its own requirements on what state a command buffer **must** be in, which are detailed in the valid usage constraints for that command.

Resetting a command buffer is an operation that discards any previously recorded commands and puts a command buffer in the *initial state*. Resetting occurs as a result of [vkResetCommandBuffer](#) or [vkResetCommandPool](#), or as part of [vkBeginCommandBuffer](#) (which additionally puts the command buffer in the *recording state*).

Secondary command buffers can be recorded to a primary command buffer via `vkCmdExecuteCommands`. This partially ties the lifecycle of the two command buffers together - if the primary is submitted to a queue, both the primary and any secondaries recorded to it move to the *pending state*. Once execution of the primary completes, so it does for any secondary recorded within it. After all executions of each command buffer complete, they each move to their appropriate completion state (either to the *executable state* or the *invalid state*, as specified above).

If a secondary moves to the *invalid state* or the *initial state*, then all primary buffers it is recorded in move to the *invalid state*. A primary moving to any other state does not affect the state of a secondary recorded in it.

NOTE

Resetting or freeing a primary command buffer removes the lifecycle linkage to all secondary command buffers that were recorded into it.

6.2. Command Pools

Command pools are opaque objects that command buffer memory is allocated from, and which allow the implementation to amortize the cost of resource creation across multiple command buffers. Command pools are externally synchronized, meaning that a command pool **must** not be used concurrently in multiple threads. That includes use via recording commands on any command buffers allocated from the pool, as well as operations that allocate, free, and reset command buffers or the pool itself.

Command pools **cannot** be destroyed or trimmed [SCID-4].

If `VkPhysicalDeviceVulkanSC10Properties::deviceDestroyFreesMemory` is `VK_TRUE`, then the memory used by command pools is returned to the system when the device is destroyed. Otherwise, it **may** not be returned to the system until the process is terminated.

Command pools are represented by `VkCommandPool` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkCommandPool)
```

To create a command pool, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateCommandPool(
    VkDevice                                device,
    const VkCommandPoolCreateInfo*          pCreateInfo,
    const VkAllocationCallbacks*           pAllocator,
    VkCommandPool*                          pCommandPool);
```

- `device` is the logical device that creates the command pool.
- `pCreateInfo` is a pointer to a `VkCommandPoolCreateInfo` structure specifying the state of the command pool object.

- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pCommandPool` is a pointer to a `VkCommandPool` handle in which the created pool is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateCommandPool` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateCommandPool-queueFamilyIndex-01937
`pCreateInfo->queueFamilyIndex` **must** be the index of a queue family available in the logical device `device`
- VUID-vkCreateCommandPool-device-05068
The number of command pools currently allocated from `device` plus 1 **must** be less than or equal to the total number of command pools requested via `VkDeviceObjectReservationCreateInfo::commandPoolRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateCommandPool-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkCreateCommandPool-pCreateInfo-parameter
`pCreateInfo` **must** be a valid pointer to a valid `VkCommandPoolCreateInfo` structure
- VUID-vkCreateCommandPool-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateCommandPool-pCommandPool-parameter
`pCommandPool` **must** be a valid pointer to a `VkCommandPool` handle
- VUID-vkCreateCommandPool-device-queuecount
The device **must** have been created with at least 1 queue

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkCommandPoolCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkCommandPoolCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkCommandPoolCreateFlags flags;
    uint32_t           queueFamilyIndex;
} VkCommandPoolCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkCommandPoolCreateFlagBits` indicating usage behavior for the pool and command buffers allocated from it.
- `queueFamilyIndex` designates a queue family as described in section [Queue Family Properties](#). All command buffers allocated from this command pool **must** be submitted on queues from the same queue family.

Valid Usage

- VUID-VkCommandPoolCreateInfo-flags-02860
If the `protectedMemory` feature is not enabled, the `VK_COMMAND_POOL_CREATE_PROTECTED_BIT` bit of `flags` **must** not be set
- VUID-VkCommandPoolCreateInfo-pNext-05002
The `pNext` chain **must** include a `VkCommandPoolMemoryReservationCreateInfo` structure

Valid Usage (Implicit)

- VUID-VkCommandPoolCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO`
- VUID-VkCommandPoolCreateInfo-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkCommandPoolMemoryReservationCreateInfo`
- VUID-VkCommandPoolCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkCommandPoolCreateInfo-flags-parameter
`flags` **must** be a valid combination of `VkCommandPoolCreateFlagBits` values

Bits which **can** be set in `VkCommandPoolCreateInfo::flags`, specifying usage behavior for a command pool, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCommandPoolCreateFlagBits {
    VK_COMMAND_POOL_CREATE_TRANSIENT_BIT = 0x00000001,
    VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT = 0x00000002,
    // Provided by VK_VERSION_1_1
    VK_COMMAND_POOL_CREATE_PROTECTED_BIT = 0x00000004,
} VkCommandPoolCreateFlagBits;
```

- **VK_COMMAND_POOL_CREATE_TRANSIENT_BIT** specifies that command buffers allocated from the pool will be short-lived, meaning that they will be reset or freed in a relatively short timeframe. This flag **may** be used by the implementation to control memory allocation behavior within the pool.
- **VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT** allows any command buffer allocated from a pool to be individually reset to the **initial state**; either by calling **vkResetCommandBuffer**, or via the implicit reset when calling **vkBeginCommandBuffer**. If this flag is not set on a pool, then **vkResetCommandBuffer** **must** not be called for any command buffer allocated from that pool.
- **VK_COMMAND_POOL_CREATE_PROTECTED_BIT** specifies that command buffers allocated from the pool are protected command buffers.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkCommandPoolCreateFlags;
```

VkCommandPoolCreateFlags is a bitmask type for setting a mask of zero or more **VkCommandPoolCreateFlagBits**.

The **pNext** chain of **VkCommandPoolCreateInfo** **must** include a **VkCommandPoolMemoryReservationCreateInfo** structure. This structure controls how much memory is allocated at command pool creation time to be used for all command buffers recorded from this pool.

The **VkCommandPoolMemoryReservationCreateInfo** structure is defined as:

```
// Provided by VKSC_VERSION_1_0
typedef struct VkCommandPoolMemoryReservationCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceSize        commandPoolReservedSize;
    uint32_t            commandPoolMaxCommandBuffers;
} VkCommandPoolMemoryReservationCreateInfo;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **commandPoolReservedSize** is the number of bytes to be allocated for all command buffer data recorded into this pool.
- **commandPoolMaxCommandBuffers** is the maximum number of command buffers that can be

allocated from this command pool.

The number of command buffers reserved using `commandPoolMaxCommandBuffers` is permanently counted against the total number of command buffers requested via `VkDeviceObjectReservationCreateInfo::commandBufferRequestCount` even if the command buffers are freed at a later time.

Each command recorded into a command buffer has an implementation-dependent size that counts against `commandPoolReservedSize`. There is no minimum command pool size, but some sizes may be too small for any commands to be recorded in them on a given implementation. Applications are expected to estimate their worst-case command buffer memory usage at development time using `vkGetCommandPoolMemoryConsumption` and reserve large enough command buffers. This command **can** also be used at runtime to verify expected memory usage.

While the memory consumption of a particular command is implementation-dependent, it is a deterministic function of the parameters to the command and of the objects used by the command (including the command buffer itself). Two command buffers will consume the same amount of pool memory if:

- all numerical parameters to each command match exactly,
- all objects used by each command are **identically defined**, and
- the order of the commands is the same.

NOTE

The rules for **identically defined** objects apply recursively, implying for example that if the command buffers are created in different devices that those devices must have been created with the same features enabled.

Each command buffer **may** require some base alignment in the pool, so the total pool memory will match if each command buffer's consumption matches and the command buffers are recorded one at a time and in the same order.

If all these criteria are satisfied, then a command pool memory consumption returned by `vkGetCommandPoolMemoryConsumption` will be sufficient to record the same command buffers again.

Valid Usage

- VUID-VkCommandPoolMemoryReservationCreateInfo-commandPoolReservedSize-05003 `commandPoolReservedSize` **must** be greater than 0
- VUID-VkCommandPoolMemoryReservationCreateInfo-commandPoolMaxCommandBuffers-05004 `commandPoolMaxCommandBuffers` **must** be greater than 0
- VUID-VkCommandPoolMemoryReservationCreateInfo-commandPoolMaxCommandBuffers-05090 `commandPoolMaxCommandBuffers` **must** be less than or equal to `VkPhysicalDeviceVulkanSC10Properties::maxCommandPoolCommandBuffers`
- VUID-VkCommandPoolMemoryReservationCreateInfo-

commandPoolMaxCommandBuffers-05074

The number of command buffers reserved by all command pools plus **commandPoolMaxCommandBuffers** **must** be less than or equal to the total number of command buffers requested via **VkDeviceObjectReservationCreateInfo::commandBufferRequestCount**

Valid Usage (Implicit)

- VUID-VkCommandPoolMemoryReservationCreateInfo-sType-sType **sType** **must** be **VK_STRUCTURE_TYPE_COMMAND_POOL_MEMORY_RESERVATION_CREATE_INFO**

To get memory usage information for a command pool object, call:

```
// Provided by VKSC_VERSION_1_0
void vkGetCommandPoolMemoryConsumption(
    VkDevice                device,
    VkCommandPool           commandPool,
    VkCommandBuffer         commandBuffer,
    VkCommandPoolMemoryConsumption* pConsumption);
```

- **device** is the logical device that owns the command pool.
- **commandPool** is the command pool from which to query the memory usage.
- **commandBuffer** is an optional command buffer from which to query the memory usage.
- **pConsumption** is a pointer to a **VkCommandPoolMemoryConsumption** structure where the memory usage is written.

Valid Usage (Implicit)

- VUID-vkGetCommandPoolMemoryConsumption-device-parameter **device** **must** be a valid **VkDevice** handle
- VUID-vkGetCommandPoolMemoryConsumption-commandPool-parameter **commandPool** **must** be a valid **VkCommandPool** handle
- VUID-vkGetCommandPoolMemoryConsumption-commandBuffer-parameter
If **commandBuffer** is not **NULL**, **commandBuffer** **must** be a valid **VkCommandBuffer** handle
- VUID-vkGetCommandPoolMemoryConsumption-pConsumption-parameter
pConsumption **must** be a valid pointer to a **VkCommandPoolMemoryConsumption** structure
- VUID-vkGetCommandPoolMemoryConsumption-commandPool-parent
commandPool **must** have been created, allocated, or retrieved from **device**
- VUID-vkGetCommandPoolMemoryConsumption-commandBuffer-parent
If **commandBuffer** is a valid handle, it **must** have been created, allocated, or retrieved from **commandPool**

Host Synchronization

- Host access to **commandPool** **must** be externally synchronized
- Host access to **commandBuffer** **must** be externally synchronized

The **VkCommandPoolMemoryConsumption** structure is defined as:

```
// Provided by VKSC_VERSION_1_0
typedef struct VkCommandPoolMemoryConsumption {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize        commandPoolAllocated;
    VkDeviceSize        commandPoolReservedSize;
    VkDeviceSize        commandBufferAllocated;
} VkCommandPoolMemoryConsumption;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **commandPoolAllocated** is the number of bytes currently allocated from this pool for command buffer data.
- **commandPoolReservedSize** is the total number of bytes available for all command buffer data recorded into this pool. This is equal to the value requested in **VkCommandPoolMemoryReservationCreateInfo::commandPoolReservedSize**.
- **commandBufferAllocated** is the number of bytes currently allocated from this pool for the specified command buffer's data. This number will be less than or equal to **VkPhysicalDeviceVulkanSC10Properties::maxCommandBufferSize**. If no command buffer is specified, then **commandBufferAllocated** will be zero.

Valid Usage (Implicit)

- VUID-VkCommandPoolMemoryConsumption-sType-sType
sType must be **VK_STRUCTURE_TYPE_COMMAND_POOL_MEMORY_CONSUMPTION**
- VUID-VkCommandPoolMemoryConsumption-pNext-pNext
pNext must be **NULL**

To reset a command pool, call:

```
// Provided by VK_VERSION_1_0
VkResult vkResetCommandPool(
    VkDevice                device,
    VkCommandPool            commandPool,
    VkCommandPoolResetFlags flags);
```

- **device** is the logical device that owns the command pool.
- **commandPool** is the command pool to reset.
- **flags** is a bitmask of **VkCommandPoolResetFlagBits** controlling the reset operation.

Resetting a command pool recycles all of the resources from all of the command buffers allocated from the command pool back to the command pool. All command buffers that have been allocated from the command pool are put in the **initial state**.

Any primary command buffer allocated from another **VkCommandPool** that is in the **recording or executable state** and has a secondary command buffer allocated from **commandPool** recorded into it, becomes **invalid**.

Valid Usage

- VUID-vkResetCommandPool-commandPool-00040
All **VkCommandBuffer** objects allocated from **commandPool** **must** not be in the **pending state**

Valid Usage (Implicit)

- VUID-vkResetCommandPool-device-parameter
device **must** be a valid **VkDevice** handle
- VUID-vkResetCommandPool-commandPool-parameter
commandPool **must** be a valid **VkCommandPool** handle
- VUID-vkResetCommandPool-flags-zero-bitmask
flags **must** be 0
- VUID-vkResetCommandPool-commandPool-parent
commandPool **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to **commandPool** **must** be externally synchronized

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

Bits which **can** be set in `vkResetCommandPool::flags`, controlling the reset operation, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCommandPoolResetFlagBits {
} VkCommandPoolResetFlagBits;
```

- `VK_COMMAND_POOL_RESET_RELEASE_RESOURCES_BIT` is not supported in Vulkan SC [SCID-4].

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkCommandPoolResetFlags;
```

`VkCommandPoolResetFlags` is a bitmask type for setting a mask of zero or more `VkCommandPoolResetFlagBits`.

6.3. Command Buffer Allocation and Management

To allocate command buffers, call:

```
// Provided by VK_VERSION_1_0
VkResult vkAllocateCommandBuffers(
    VkDevice device,
    const VkCommandBufferAllocateInfo* pAllocateInfo,
    VkCommandBuffer* pCommandBuffers);
```

- `device` is the logical device that owns the command pool.
- `pAllocateInfo` is a pointer to a `VkCommandBufferAllocateInfo` structure describing parameters of the allocation. `commandPool` **may** be accessed any time one of the resulting command buffers is accessed.
- `pCommandBuffers` is a pointer to an array of `VkCommandBuffer` handles in which the resulting command buffer objects are returned. The array **must** be at least the length specified by the `commandBufferCount` member of `pAllocateInfo`. Each allocated command buffer begins in the initial state.

`vkAllocateCommandBuffers` **can** be used to allocate multiple command buffers. If the allocation of any of those command buffers fails, the implementation **must** free all successfully allocated command buffer objects from this command, set all entries of the `pCommandBuffers` array to `NULL` and return the error.

NOTE

Filling `pCommandBuffers` with `NULL` values on failure is an exception to the default error behavior that output parameters will have undefined contents.

When command buffers are first allocated, they are in the `initial state`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkAllocateCommandBuffers` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkAllocateCommandBuffers-device-parameter **device** **must** be a valid [VkDevice](#) handle
- VUID-vkAllocateCommandBuffers-pAllocateInfo-parameter **pAllocateInfo** **must** be a valid pointer to a valid [VkCommandBufferAllocateInfo](#) structure
- VUID-vkAllocateCommandBuffers-pCommandBuffers-parameter **pCommandBuffers** **must** be a valid pointer to an array of **pAllocateInfo->commandBufferCount** [VkCommandBuffer](#) handles
- VUID-vkAllocateCommandBuffers-device-queuecount
The device **must** have been created with at least **1** queue
- VUID-vkAllocateCommandBuffers-pAllocateInfo::commandBufferCount-arraylength
pAllocateInfo->commandBufferCount **must** be greater than **0**

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The [VkCommandBufferAllocateInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkCommandBufferAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkCommandPool       commandPool;
    VkCommandBufferLevel level;
    uint32_t            commandBufferCount;
} VkCommandBufferAllocateInfo;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **commandPool** is the command pool from which the command buffers are allocated.
- **level** is a [VkCommandBufferLevel](#) value specifying the command buffer level.
- **commandBufferCount** is the number of command buffers to allocate from the pool.

The number of command buffers allocated using `commandBufferCount` counts against the maximum number of command buffers reserved via `VkCommandPoolMemoryReservationCreateInfo::commandPoolMaxCommandBuffers` specified when `commandPool` was created. Once command buffers are freed with `vkFreeCommandBuffers`, they can be allocated from `commandPool` again.

Valid Usage

- VUID-VkCommandBufferAllocateInfo-level-09660
If `level` is not `VK_COMMAND_BUFFER_LEVEL_PRIMARY`, `commandPool` **must** have been allocated with a `queueFamilyIndex` identifying a queue family that supports one of the `VK_QUEUE_TRANSFER_BIT`, `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` capabilities
- VUID-VkCommandBufferAllocateInfo-commandPool-05006
The number of command buffers currently allocated from `commandPool` plus `commandBufferCount` **must** be less than or equal to the value of `VkCommandPoolMemoryReservationCreateInfo::commandPoolMaxCommandBuffers` specified when `commandPool` was created

Valid Usage (Implicit)

- VUID-VkCommandBufferAllocateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO`
- VUID-VkCommandBufferAllocateInfo-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkCommandBufferAllocateInfo-commandPool-parameter
`commandPool` **must** be a valid `VkCommandPool` handle
- VUID-VkCommandBufferAllocateInfo-level-parameter
`level` **must** be a valid `VkCommandBufferLevel` value

Host Synchronization

- Host access to `commandPool` **must** be externally synchronized

Possible values of `VkCommandBufferAllocateInfo::level`, specifying the command buffer level, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCommandBufferLevel {
    VK_COMMAND_BUFFER_LEVEL_PRIMARY = 0,
    VK_COMMAND_BUFFER_LEVEL_SECONDARY = 1,
} VkCommandBufferLevel;
```

- `VK_COMMAND_BUFFER_LEVEL_PRIMARY` specifies a primary command buffer.
- `VK_COMMAND_BUFFER_LEVEL_SECONDARY` specifies a secondary command buffer.

To reset a command buffer, call:

```
// Provided by VK_VERSION_1_0
VkResult vkResetCommandBuffer(
    VkCommandBuffer          commandBuffer,
    VkCommandBufferResetFlags flags);
```

- **commandBuffer** is the command buffer to reset. The command buffer **can** be in any state other than **pending**, and is moved into the **initial state**.
- **flags** is a bitmask of **VkCommandBufferResetFlagBits** controlling the reset operation.

Any primary command buffer that is in the **recording or executable state** and has **commandBuffer** recorded into it, becomes **invalid**.

After a command buffer is reset, any objects or memory specified by commands recorded into the command buffer **must** no longer be accessed when the command buffer is accessed by the implementation.

Valid Usage

- VUID-vkResetCommandBuffer-commandBuffer-00045
commandBuffer **must** not be in the **pending state**
- VUID-vkResetCommandBuffer-commandBuffer-00046
commandBuffer **must** have been allocated from a pool that was created with the **VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT**
- VUID-vkResetCommandBuffer-commandPoolResetCommandBuffer-05135
commandPoolResetCommandBuffer **must** be supported

Valid Usage (Implicit)

- VUID-vkResetCommandBuffer-commandBuffer-parameter
commandBuffer **must** be a valid **VkCommandBuffer** handle
- VUID-vkResetCommandBuffer-flags-parameter
flags **must** be a valid combination of **VkCommandBufferResetFlagBits** values

Host Synchronization

- Host access to **commandBuffer** **must** be externally synchronized
- Host access to the **VkCommandPool** that **commandBuffer** was allocated from **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

Bits which **can** be set in `vkResetCommandBuffer::flags`, controlling the reset operation, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCommandBufferResetFlagBits {
    VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT = 0x00000001,
} VkCommandBufferResetFlagBits;
```

- `VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT` specifies that most or all memory resources currently owned by the command buffer **should** be returned to the parent command pool. If this flag is not set, then the command buffer **may** hold onto memory resources and reuse them when recording commands. `commandBuffer` is moved to the [initial state](#).

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkCommandBufferResetFlags;
```

`VkCommandBufferResetFlags` is a bitmask type for setting a mask of zero or more `VkCommandBufferResetFlagBits`.

To free command buffers, call:

```
// Provided by VK_VERSION_1_0
void vkFreeCommandBuffers(
    VkDevice          device,
    VkCommandPool     commandPool,
    uint32_t          commandBufferCount,
    const VkCommandBuffer* pCommandBuffers);
```

- `device` is the logical device that owns the command pool.
- `commandPool` is the command pool from which the command buffers were allocated.
- `commandBufferCount` is the length of the `pCommandBuffers` array.
- `pCommandBuffers` is a pointer to an array of handles of command buffers to free.

Any primary command buffer that is in the [recording or executable state](#) and has any element of

`pCommandBuffers` recorded into it, becomes `invalid`.

Freeing a command buffer does not return the memory used by command recording back to its parent command pool. This memory will be reclaimed the next time `vkResetCommandPool` is called.

Valid Usage

- VUID-vkFreeCommandBuffers-pCommandBuffers-00047
All elements of `pCommandBuffers` **must** not be in the `pending state`
- VUID-vkFreeCommandBuffers-pCommandBuffers-00048
`pCommandBuffers` **must** be a valid pointer to an array of `commandBufferCount` `VkCommandBuffer` handles, each element of which **must** either be a valid handle or `NULL`

Valid Usage (Implicit)

- VUID-vkFreeCommandBuffers-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkFreeCommandBuffers-commandPool-parameter
`commandPool` **must** be a valid `VkCommandPool` handle
- VUID-vkFreeCommandBuffers-commandBufferCount-arraylength
`commandBufferCount` **must** be greater than 0
- VUID-vkFreeCommandBuffers-commandPool-parent
`commandPool` **must** have been created, allocated, or retrieved from `device`
- VUID-vkFreeCommandBuffers-pCommandBuffers-parent
Each element of `pCommandBuffers` that is a valid handle **must** have been created, allocated, or retrieved from `commandPool`

Host Synchronization

- Host access to `commandPool` **must** be externally synchronized
- Host access to each member of `pCommandBuffers` **must** be externally synchronized

6.4. Command Buffer Recording

To begin recording a command buffer, call:

```
// Provided by VK_VERSION_1_0
VkResult vkBeginCommandBuffer(
    VkCommandBuffer                                commandBuffer,
    const VkCommandBufferBeginInfo*                pBeginInfo);
```

- `commandBuffer` is the handle of the command buffer which is to be put in the recording state.
- `pBeginInfo` is a pointer to a `VkCommandBufferBeginInfo` structure defining additional information about how the command buffer begins recording.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkBeginCommandBuffer` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkBeginCommandBuffer-commandBuffer-00049
`commandBuffer` **must** not be in the `recording or pending state`
- VUID-vkBeginCommandBuffer-commandBuffer-00050
If `commandBuffer` was allocated from a `VkCommandPool` which did not have the `VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT` flag set, `commandBuffer` **must** be in the `initial state`
- VUID-vkBeginCommandBuffer-commandPoolResetCommandBuffer-05136
If `commandPoolResetCommandBuffer` is not supported, `commandBuffer` **must** be in the `initial state`
- VUID-vkBeginCommandBuffer-commandBuffer-00051
If `commandBuffer` is a secondary command buffer, the `pInheritanceInfo` member of `pBeginInfo` **must** be a valid `VkCommandBufferInheritanceInfo` structure
- VUID-vkBeginCommandBuffer-commandBuffer-00052
If `commandBuffer` is a secondary command buffer and either the `occlusionQueryEnable` member of the `pInheritanceInfo` member of `pBeginInfo` is `VK_FALSE`, or the `occlusionQueryPrecise` feature is not enabled, then `pBeginInfo->pInheritanceInfo->queryFlags` **must** not contain `VK_QUERY_CONTROL_PRECISE_BIT`
- VUID-vkBeginCommandBuffer-commandBuffer-02840
If `commandBuffer` is a primary command buffer, then `pBeginInfo->flags` **must** not set both the `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT` and the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` flags
- VUID-vkBeginCommandBuffer-commandPoolMultipleCommandBuffersRecording-05007
If `commandPoolMultipleCommandBuffersRecording` is `VK_FALSE`, then the command pool that `commandBuffer` was created from **must** have no other command buffers in the `recording state`
- VUID-vkBeginCommandBuffer-commandBufferSimultaneousUse-05008
If `commandBufferSimultaneousUse` is `VK_FALSE`, then `pBeginInfo->flags` **must** not include `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT`

Valid Usage (Implicit)

- VUID-vkBeginCommandBuffer-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkBeginCommandBuffer-pBeginInfo-parameter

`pBeginInfo` **must** be a valid pointer to a valid `VkCommandBufferBeginInfo` structure

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkCommandBufferBeginInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkCommandBufferBeginInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkCommandBufferUsageFlags flags;
    const VkCommandBufferInheritanceInfo* pInheritanceInfo;
} VkCommandBufferBeginInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkCommandBufferUsageFlagBits` specifying usage behavior for the command buffer.
- `pInheritanceInfo` is a pointer to a `VkCommandBufferInheritanceInfo` structure, used if `commandBuffer` is a secondary command buffer. If this is a primary command buffer, then this value is ignored.

Valid Usage

- VUID-VkCommandBufferBeginInfo-flags-09123
If `flags` contains `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT`, the `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations

- VUID-VkCommandBufferBeginInfo-flags-05009
If `flags` contains `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT` and `secondaryCommandBufferNullOrImagelessFramebuffer` is `VK_TRUE`, the `framebuffer` member of `pInheritanceInfo` **must** be either `VK_NULL_HANDLE`, or a valid `VkFramebuffer` that is compatible with the `renderPass` member of `pInheritanceInfo`
- VUID-VkCommandBufferBeginInfo-flags-05010
If `flags` contains `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT` and `secondaryCommandBufferNullOrImagelessFramebuffer` is `VK_FALSE`, the `framebuffer` member of `pInheritanceInfo` **must** be a valid `VkFramebuffer` that is compatible with the `renderPass` member of `pInheritanceInfo` and **must** not have been created with a `VkFramebufferCreateInfo::flags` value that includes `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`
- VUID-VkCommandBufferBeginInfo-flags-06000
If `flags` contains `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT` the `renderPass` member of `pInheritanceInfo` **must** be a valid `VkRenderPass`
- VUID-VkCommandBufferBeginInfo-flags-06001
If `flags` contains `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT` the `subpass` member of `pInheritanceInfo` **must** be a valid subpass index within the `renderPass` member of `pInheritanceInfo`

Valid Usage (Implicit)

- VUID-VkCommandBufferBeginInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO`
- VUID-VkCommandBufferBeginInfo-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkDeviceGroupCommandBufferBeginInfo`
- VUID-VkCommandBufferBeginInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkCommandBufferBeginInfo-flags-parameter
`flags` **must** be a valid combination of `VkCommandBufferUsageFlagBits` values

Bits which **can** be set in `VkCommandBufferBeginInfo::flags`, specifying usage behavior for a command buffer, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCommandBufferUsageFlagBits {
    VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT = 0x00000001,
    VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT = 0x00000002,
    VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT = 0x00000004,
} VkCommandBufferUsageFlagBits;
```

- `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT` specifies that each recording of the command buffer will only be submitted once, and the command buffer will be reset and recorded again

between each submission.

- `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT` specifies that a secondary command buffer is considered to be entirely inside a render pass. If this is a primary command buffer, then this bit is ignored.
- `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` specifies that a command buffer **can** be resubmitted to any queue of the same queue family while it is in the *pending state*, and recorded into multiple primary command buffers.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkCommandBufferUsageFlags;
```

`VkCommandBufferUsageFlags` is a bitmask type for setting a mask of zero or more `VkCommandBufferUsageFlagBits`.

If the command buffer is a secondary command buffer, then the `VkCommandBufferInheritanceInfo` structure defines any state that will be inherited from the primary command buffer:

```
// Provided by VK_VERSION_1_0
typedef struct VkCommandBufferInheritanceInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkRenderPass        renderPass;
    uint32_t            subpass;
    VkFramebuffer       framebuffer;
    VkBool32            occlusionQueryEnable;
    VkQueryControlFlags queryFlags;
    VkQueryPipelineStatisticFlags pipelineStatistics;
} VkCommandBufferInheritanceInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `renderPass` is a `VkRenderPass` object defining which render passes the `VkCommandBuffer` will be **compatible** with and **can** be executed within.
- `subpass` is the index of the subpass within the render pass instance that the `VkCommandBuffer` will be executed within.
- `framebuffer` **can** refer to the `VkFramebuffer` object that the `VkCommandBuffer` will be rendering to if it is executed within a render pass instance. It **can** be `VK_NULL_HANDLE` if the framebuffer is not known.

NOTE

Specifying the exact framebuffer that the secondary command buffer will be executed with **may** result in better performance at command buffer execution time.

- `occlusionQueryEnable` specifies whether the command buffer **can** be executed while an occlusion query is active in the primary command buffer. If this is `VK_TRUE`, then this command

buffer **can** be executed whether the primary command buffer has an occlusion query active or not. If this is `VK_FALSE`, then the primary command buffer **must** not have an occlusion query active.

- `queryFlags` specifies the query flags that **can** be used by an active occlusion query in the primary command buffer when this secondary command buffer is executed. If this value includes the `VK_QUERY_CONTROL_PRECISE_BIT` bit, then the active query **can** return boolean results or actual sample counts. If this bit is not set, then the active query **must** not use the `VK_QUERY_CONTROL_PRECISE_BIT` bit.
- `pipelineStatistics` is a bitmask of `VkQueryPipelineStatisticFlagBits` specifying the set of pipeline statistics that **can** be counted by an active query in the primary command buffer when this secondary command buffer is executed. If this value includes a given bit, then this command buffer **can** be executed whether the primary command buffer has a pipeline statistics query active that includes this bit or not. If this value excludes a given bit, then the active pipeline statistics query **must** not be from a query pool that counts that statistic.

If the `VkCommandBuffer` will not be executed within a render pass instance, `renderPass`, `subpass`, and `framebuffer` are ignored.

Valid Usage

- VUID-VkCommandBufferInheritanceInfo-occlusionQueryEnable-00056
If the `inheritedQueries` feature is not enabled, `occlusionQueryEnable` **must** be `VK_FALSE`
- VUID-VkCommandBufferInheritanceInfo-queryFlags-00057
If the `inheritedQueries` feature is enabled, `queryFlags` **must** be a valid combination of `VkQueryControlFlagBits` values
- VUID-VkCommandBufferInheritanceInfo-queryFlags-02788
If the `inheritedQueries` feature is not enabled, `queryFlags` **must** be `0`
- VUID-VkCommandBufferInheritanceInfo-pipelineStatistics-02789
If the `pipelineStatisticsQuery` feature is enabled, `pipelineStatistics` **must** be a valid combination of `VkQueryPipelineStatisticFlagBits` values
- VUID-VkCommandBufferInheritanceInfo-pipelineStatistics-00058
If the `pipelineStatisticsQuery` feature is not enabled, `pipelineStatistics` **must** be `0`

Valid Usage (Implicit)

- VUID-VkCommandBufferInheritanceInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_INFO`
- VUID-VkCommandBufferInheritanceInfo-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkCommandBufferInheritanceInfo-commonparent
Both of `framebuffer`, and `renderPass` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same `VkDevice`

NOTE

On some implementations, not using the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` bit enables command buffers to be patched in-place if needed, rather than creating a copy of the command buffer.

If a command buffer is in the [invalid, or executable state](#), and the command buffer was allocated from a command pool with the `VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT` flag set, then `vkBeginCommandBuffer` implicitly resets the command buffer, behaving as if `vkResetCommandBuffer` had been called with `VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT` not set. After the implicit reset, `commandBuffer` is moved to the [recording state](#).

Once recording starts, an application records a sequence of commands (`vkCmd*`) to set state in the command buffer, draw, dispatch, and other commands.

To complete recording of a command buffer, call:

```
// Provided by VK_VERSION_1_0
VkResult vkEndCommandBuffer(
    VkCommandBuffer                commandBuffer);
```

- `commandBuffer` is the command buffer to complete recording.

The command buffer **must** have been in the [recording state](#), and, if successful, is moved to the [executable state](#).

If there was an error during recording, the application will be notified by an unsuccessful return code returned by `vkEndCommandBuffer`, and the command buffer will be moved to the [invalid state](#).

If recording a command would exceed the amount of command pool memory reserved by `VkCommandPoolMemoryReservationCreateInfo::commandPoolReservedSize` or the maximum supported command buffer size reported in `VkPhysicalDeviceVulkanSC10Properties::maxCommandBufferSize`, the implementation **may** report a `VK_FAULT_TYPE_COMMAND_BUFFER_FULL` fault. The command buffer remains in the [recording state](#) until `vkEndCommandBuffer` is called. When `vkEndCommandBuffer` is called on a command buffer for which the command pool memory reservation was exceeded during recording, it **must** return `VK_ERROR_OUT_OF_DEVICE_MEMORY`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkEndCommandBuffer` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkEndCommandBuffer-commandBuffer-00059
`commandBuffer` **must** be in the [recording state](#)
- VUID-vkEndCommandBuffer-commandBuffer-00060
If `commandBuffer` is a primary command buffer, there **must** not be an active render pass instance
- VUID-vkEndCommandBuffer-commandBuffer-00061
All queries made [active](#) during the recording of `commandBuffer` **must** have been made

inactive

- VUID-vkEndCommandBuffer-commandBuffer-01815

If `commandBuffer` is a secondary command buffer, there **must** not be an outstanding `vkCmdBeginDebugUtilsLabelEXT` command recorded to `commandBuffer` that has not previously been ended by a call to `vkCmdEndDebugUtilsLabelEXT`

Valid Usage (Implicit)

- VUID-vkEndCommandBuffer-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

When a command buffer is in the executable state, it **can** be submitted to a queue for execution.

6.5. Command Buffer Submission

NOTE

Submission can be a high overhead operation, and applications **should** attempt to batch work together into as few calls to `vkQueueSubmit` or `vkQueueSubmit2KHR` as possible.

To submit command buffers to a queue, call:

```
// Provided by VK_KHR_synchronization2
VkResult vkQueueSubmit2KHR(
    VkQueue          queue,
    uint32_t         submitCount,
    const VkSubmitInfo2* pSubmits,
    VkFence          fence);
```

- `queue` is the queue that the command buffers will be submitted to.
- `submitCount` is the number of elements in the `pSubmits` array.
- `pSubmits` is a pointer to an array of `VkSubmitInfo2` structures, each specifying a command buffer submission batch. Command buffers and semaphores specified in this array **may** be accessed at any point until the [queue operations](#) they define complete execution on the device.
- `fence` is an **optional** handle to a fence to be signaled once all submitted command buffers have completed execution. If `fence` is not `VK_NULL_HANDLE`, it defines a [fence signal operation](#). If it is not `VK_NULL_HANDLE`, `fence` **may** be accessed at any point until this command completes on the device.

`vkQueueSubmit2KHR` is a [queue submission command](#), with each batch defined by an element of `pSubmits`.

Semaphore operations submitted with `vkQueueSubmit2KHR` have additional ordering constraints compared to other submission commands, with dependencies involving previous and subsequent queue operations. Information about these additional constraints can be found in the [semaphore](#) section of [the synchronization chapter](#).

If any command buffer submitted to this queue is in the [executable state](#), it is moved to the [pending state](#). Once execution of all submissions of a command buffer complete, it moves from the [pending state](#), back to the [executable state](#). If a command buffer was recorded with the `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT` flag, it instead moves back to the [invalid state](#).

If `vkQueueSubmit2KHR` fails, it **may** return `VK_ERROR_OUT_OF_HOST_MEMORY` or `VK_ERROR_OUT_OF_DEVICE_MEMORY`. If it does, the implementation **must** ensure that the state and contents of any resources or synchronization primitives referenced by the submitted command buffers and any semaphores referenced by `pSubmits` is unaffected by the call or its failure. If `vkQueueSubmit2KHR` fails in such a way that the implementation is unable to make that guarantee, the implementation **must** return `VK_ERROR_DEVICE_LOST`. See [Lost Device](#).

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkQueueSubmit2KHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkQueueSubmit2-fence-04894
If `fence` is not `VK_NULL_HANDLE`, `fence` **must** be unsignaled
- VUID-vkQueueSubmit2-fence-04895
If `fence` is not `VK_NULL_HANDLE`, `fence` **must** not be associated with any other queue

command that has not yet completed execution on that queue

- VUID-vkQueueSubmit2-synchronization2-03866
The **synchronization2** feature **must** be enabled
- VUID-vkQueueSubmit2-commandBuffer-03867
If a command recorded into the **commandBuffer** member of any element of the **pCommandBufferInfos** member of any element of **pSubmits** referenced a **VkEvent**, that event **must** not be referenced by a command that has been submitted to another queue and is still in the *pending state*
- VUID-vkQueueSubmit2-semaphore-03868
The **semaphore** member of any binary semaphore element of the **pSignalSemaphoreInfos** member of any element of **pSubmits** **must** be unsignaled when the semaphore signal operation it defines is executed on the device
- VUID-vkQueueSubmit2-stageMask-03869
The **stageMask** member of any element of the **pSignalSemaphoreInfos** member of any element of **pSubmits** **must** only include pipeline stages that are supported by the queue family which **queue** belongs to
- VUID-vkQueueSubmit2-stageMask-03870
The **stageMask** member of any element of the **pWaitSemaphoreInfos** member of any element of **pSubmits** **must** only include pipeline stages that are supported by the queue family which **queue** belongs to
- VUID-vkQueueSubmit2-semaphore-03871
When a semaphore wait operation for a binary semaphore is executed, as defined by the **semaphore** member of any element of the **pWaitSemaphoreInfos** member of any element of **pSubmits**, there **must** be no other queues waiting on the same semaphore
- VUID-vkQueueSubmit2-semaphore-03873
The **semaphore** member of any element of the **pWaitSemaphoreInfos** member of any element of **pSubmits** that was created with a **VkSemaphoreType** of **VK_SEMAPHORE_TYPE_BINARY** **must** reference a semaphore signal operation that has been submitted for execution and any **semaphore signal operations** on which it depends **must** have also been submitted for execution
- VUID-vkQueueSubmit2-commandBuffer-03874
The **commandBuffer** member of any element of the **pCommandBufferInfos** member of any element of **pSubmits** **must** be in the **pending or executable state**
- VUID-vkQueueSubmit2-commandBuffer-03875
If a command recorded into the **commandBuffer** member of any element of the **pCommandBufferInfos** member of any element of **pSubmits** was not recorded with the **VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT**, it **must** not be in the **pending state**
- VUID-vkQueueSubmit2-commandBuffer-03876
Any **secondary command buffers recorded** into the **commandBuffer** member of any element of the **pCommandBufferInfos** member of any element of **pSubmits** **must** be in the **pending or executable state**
- VUID-vkQueueSubmit2-commandBuffer-03877
If any **secondary command buffers recorded** into the **commandBuffer** member of any

element of the `pCommandBufferInfos` member of any element of `pSubmits` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT`, it **must** not be in the `pending` state

- VUID-vkQueueSubmit2-commandBuffer-03878

The `commandBuffer` member of any element of the `pCommandBufferInfos` member of any element of `pSubmits` **must** have been allocated from a `VkCommandPool` that was created for the same queue family `queue` belongs to

- VUID-vkQueueSubmit2-commandBuffer-03879

If a command recorded into the `commandBuffer` member of any element of the `pCommandBufferInfos` member of any element of `pSubmits` includes a `Queue Family Ownership Transfer Acquire Operation`, there **must** exist a previously submitted `Queue Family Ownership Transfer Release Operation` on a queue in the queue family identified by the acquire operation, with parameters matching the acquire operation as defined in the definition of such `acquire operations`, and which happens before the acquire operation

- VUID-vkQueueSubmit2-commandBuffer-10910

If a command recorded into the `commandBuffer` member of any element of the `pCommandBufferInfos` member of any element of `pSubmits` includes a `Queue Family Ownership Transfer Acquire Operation`, the affected resource **must** not be modified in any way between the last matching release operation and the acquire operation

- VUID-vkQueueSubmit2-commandBuffer-03880

If a command recorded into the `commandBuffer` member of any element of the `pCommandBufferInfos` member of any element of `pSubmits` was a `vkCmdBeginQuery` whose `queryPool` was created with a `queryType` of `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, the `profiling lock` **must** have been held continuously on the `VkDevice` that `queue` was retrieved from, throughout recording of those command buffers

- VUID-vkQueueSubmit2-queue-06447

If `queue` was not created with `VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT`, the `flags` member of any element of `pSubmits` **must** not include `VK_SUBMIT_PROTECTED_BIT_KHR`

Valid Usage (Implicit)

- VUID-vkQueueSubmit2-queue-parameter

`queue` **must** be a valid `VkQueue` handle

- VUID-vkQueueSubmit2-pSubmits-parameter

If `submitCount` is not 0, `pSubmits` **must** be a valid pointer to an array of `submitCount` valid `VkSubmitInfo2` structures

- VUID-vkQueueSubmit2-fence-parameter

If `fence` is not `VK_NULL_HANDLE`, `fence` **must** be a valid `VkFence` handle

- VUID-vkQueueSubmit2-commonparent

Both of `fence`, and `queue` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to **queue** **must** be externally synchronized
- Host access to **fence** **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
-	-	Any	-

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_DEVICE_LOST**
- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The **VkSubmitInfo2** structure is defined as:

```
typedef struct VkSubmitInfo2 {  
    VkStructureType      sType;  
    const void*          pNext;  
    VkSubmitFlags        flags;  
    uint32_t             waitSemaphoreInfoCount;  
    const VkSemaphoreSubmitInfo* pWaitSemaphoreInfos;  
    uint32_t             commandBufferInfoCount;  
    const VkCommandBufferSubmitInfo* pCommandBufferInfos;  
    uint32_t             signalSemaphoreInfoCount;  
    const VkSemaphoreSubmitInfo* pSignalSemaphoreInfos;  
} VkSubmitInfo2;
```

```
// Provided by VK_KHR_synchronization2  
typedef VkSubmitInfo2 VkSubmitInfo2KHR;
```

- **sType** is a **VkStructureType** value identifying this structure.

- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkSubmitFlagBits`.
- `waitSemaphoreInfoCount` is the number of elements in `pWaitSemaphoreInfos`.
- `pWaitSemaphoreInfos` is a pointer to an array of `VkSemaphoreSubmitInfo` structures defining semaphore wait operations.
- `commandBufferInfoCount` is the number of elements in `pCommandBufferInfos` and the number of command buffers to execute in the batch.
- `pCommandBufferInfos` is a pointer to an array of `VkCommandBufferSubmitInfo` structures describing command buffers to execute in the batch.
- `signalSemaphoreInfoCount` is the number of elements in `pSignalSemaphoreInfos`.
- `pSignalSemaphoreInfos` is a pointer to an array of `VkSemaphoreSubmitInfo` describing semaphore signal operations.

Valid Usage

- VUID-VkSubmitInfo2-flags-03886
If `flags` includes `VK_SUBMIT_PROTECTED_BIT`, all elements of `pCommandBuffers` **must** be protected command buffers
- VUID-VkSubmitInfo2-flags-03887
If `flags` does not include `VK_SUBMIT_PROTECTED_BIT`, each element of `pCommandBuffers` **must** not be a protected command buffer

Valid Usage (Implicit)

- VUID-VkSubmitInfo2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SUBMIT_INFO_2`
- VUID-VkSubmitInfo2-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkPerformanceQuerySubmitInfoKHR`
- VUID-VkSubmitInfo2-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkSubmitInfo2-flags-parameter
`flags` **must** be a valid combination of `VkSubmitFlagBits` values
- VUID-VkSubmitInfo2-pWaitSemaphoreInfos-parameter
If `waitSemaphoreInfoCount` is not 0, `pWaitSemaphoreInfos` **must** be a valid pointer to an array of `waitSemaphoreInfoCount` valid `VkSemaphoreSubmitInfo` structures
- VUID-VkSubmitInfo2-pCommandBufferInfos-parameter
If `commandBufferInfoCount` is not 0, `pCommandBufferInfos` **must** be a valid pointer to an array of `commandBufferInfoCount` valid `VkCommandBufferSubmitInfo` structures
- VUID-VkSubmitInfo2-pSignalSemaphoreInfos-parameter
If `signalSemaphoreInfoCount` is not 0, `pSignalSemaphoreInfos` **must** be a valid pointer to an

array of `signalSemaphoreInfoCount` valid `VkSemaphoreSubmitInfo` structures

Bits which **can** be set in `VkSubmitInfo2::flags`, specifying submission behavior, are:

```
typedef enum VkSubmitFlagBits {  
    VK_SUBMIT_PROTECTED_BIT = 0x00000001,  
    // Provided by VK_KHR_synchronization2  
    VK_SUBMIT_PROTECTED_BIT_KHR = VK_SUBMIT_PROTECTED_BIT,  
} VkSubmitFlagBits;
```

```
// Provided by VK_KHR_synchronization2  
typedef VkSubmitFlagBits VkSubmitFlagBitsKHR;
```

- `VK_SUBMIT_PROTECTED_BIT` specifies that this batch is a protected submission.

```
typedef VkFlags VkSubmitFlags;
```

```
// Provided by VK_KHR_synchronization2  
typedef VkSubmitFlags VkSubmitFlagsKHR;
```

`VkSubmitFlags` is a bitmask type for setting a mask of zero or more `VkSubmitFlagBits`.

The `VkSemaphoreSubmitInfo` structure is defined as:

```
typedef struct VkSemaphoreSubmitInfo {  
    VkStructureType      sType;  
    const void*          pNext;  
    VkSemaphore          semaphore;  
    uint64_t             value;  
    VkPipelineStageFlags2 stageMask;  
    uint32_t             deviceIndex;  
} VkSemaphoreSubmitInfo;
```

```
// Provided by VK_KHR_synchronization2  
typedef VkSemaphoreSubmitInfo VkSemaphoreSubmitInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `semaphore` is a `VkSemaphore` affected by this operation.
- `value` is ignored.
- `stageMask` is a `VkPipelineStageFlags2` mask of pipeline stages which limit the first

synchronization scope of a semaphore signal operation, or second synchronization scope of a semaphore wait operation as described in the [semaphore wait operation](#) and [semaphore signal operation](#) sections of [the synchronization chapter](#).

- **deviceIndex** is the index of the device within a device group that executes the semaphore wait or signal operation.

Whether this structure defines a semaphore wait or signal operation is defined by how it is used.

Valid Usage

- VUID-VkSemaphoreSubmitInfo-stageMask-03929
If the **geometryShader** feature is not enabled, **stageMask** **must** not contain **VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT**
- VUID-VkSemaphoreSubmitInfo-stageMask-03930
If the **tessellationShader** feature is not enabled, **stageMask** **must** not contain **VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT** or **VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT**
- VUID-VkSemaphoreSubmitInfo-stageMask-07317
If the **attachmentFragmentShadingRate** feature is not enabled, **stageMask** **must** not contain **VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR**
- VUID-VkSemaphoreSubmitInfo-device-03888
If the **device** that **semaphore** was created on is not a device group, **deviceIndex** **must** be 0
- VUID-VkSemaphoreSubmitInfo-device-03889
If the **device** that **semaphore** was created on is a device group, **deviceIndex** **must** be a valid device index
- VUID-VkSemaphoreSubmitInfoKHR-semaphore-05094
If **semaphore** has a payload of **NvSciSyncObj**, **value** **must** be calculated by application via [NvSciSync APIs](#)

Valid Usage (Implicit)

- VUID-VkSemaphoreSubmitInfo-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_SEMAPHORE_SUBMIT_INFO**
- VUID-VkSemaphoreSubmitInfo-pNext-pNext
pNext **must** be **NULL**
- VUID-VkSemaphoreSubmitInfo-semaphore-parameter
semaphore **must** be a valid [VkSemaphore](#) handle
- VUID-VkSemaphoreSubmitInfo-stageMask-parameter
stageMask **must** be a valid combination of [VkPipelineStageFlagBits2](#) values

The **VkCommandBufferSubmitInfo** structure is defined as:

```
typedef struct VkCommandBufferSubmitInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkCommandBuffer    commandBuffer;
    uint32_t           deviceMask;
} VkCommandBufferSubmitInfo;
```

```
// Provided by VK_KHR_synchronization2
typedef VkCommandBufferSubmitInfo VkCommandBufferSubmitInfoKHR;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **commandBuffer** is a [VkCommandBuffer](#) to be submitted for execution.
- **deviceMask** is a bitmask indicating which devices in a device group execute the command buffer. A **deviceMask** of **0** is equivalent to setting all bits corresponding to valid devices in the group to **1**.

Valid Usage

- VUID-VkCommandBufferSubmitInfo-commandBuffer-03890
commandBuffer **must** not have been allocated with **VK_COMMAND_BUFFER_LEVEL_SECONDARY**
- VUID-VkCommandBufferSubmitInfo-deviceMask-03891
If **deviceMask** is not **0**, it **must** be a valid device mask

Valid Usage (Implicit)

- VUID-VkCommandBufferSubmitInfo-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_COMMAND_BUFFER_SUBMIT_INFO**
- VUID-VkCommandBufferSubmitInfo-pNext-pNext
pNext **must** be **NULL**
- VUID-VkCommandBufferSubmitInfo-commandBuffer-parameter
commandBuffer **must** be a valid [VkCommandBuffer](#) handle

To submit command buffers to a queue, call:

```
// Provided by VK_VERSION_1_0
VkResult vkQueueSubmit(
    VkQueue          queue,
    uint32_t         submitCount,
    const VkSubmitInfo* pSubmits,
    VkFence          fence);
```

- `queue` is the queue that the command buffers will be submitted to.
- `submitCount` is the number of elements in the `pSubmits` array.
- `pSubmits` is a pointer to an array of `VkSubmitInfo` structures, each specifying a command buffer submission batch. Command buffers and semaphores specified in this array **may** be accessed at any point until the `queue operations` they define complete execution on the device.
- `fence` is an **optional** handle to a fence to be signaled once all submitted command buffers have completed execution. If `fence` is not `VK_NULL_HANDLE`, it defines a `fence signal operation`. If it is not `VK_NULL_HANDLE`, `fence` **may** be accessed at any point until this command completes on the device.

`vkQueueSubmit` is a `queue submission command`, with each batch defined by an element of `pSubmits`. Batches begin execution in the order they appear in `pSubmits`, but **may** complete out of order.

Fence and semaphore operations submitted with `vkQueueSubmit` have additional ordering constraints compared to other submission commands, with dependencies involving previous and subsequent queue operations. Information about these additional constraints can be found in the `semaphore` and `fence` sections of [the synchronization chapter](#).

Details on the interaction of `pWaitDstStageMask` with synchronization are described in the `semaphore wait operation` section of [the synchronization chapter](#).

The order that batches appear in `pSubmits` is used to determine `submission order`, and thus all the `implicit ordering guarantees` that respect it. Other than these implicit ordering guarantees and any `explicit synchronization primitives`, these batches **may** overlap or otherwise execute out of order.

If any command buffer submitted to this queue is in the `executable state`, it is moved to the `pending state`. Once execution of all submissions of a command buffer complete, it moves from the `pending state`, back to the `executable state`. If a command buffer was recorded with the `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT` flag, it instead moves to the `invalid state`.

If `vkQueueSubmit` fails, it **may** return `VK_ERROR_OUT_OF_HOST_MEMORY` or `VK_ERROR_OUT_OF_DEVICE_MEMORY`. If it does, the implementation **must** ensure that the state and contents of any resources or synchronization primitives referenced by the submitted command buffers and any semaphores referenced by `pSubmits` is unaffected by the call or its failure. If `vkQueueSubmit` fails in such a way that the implementation is unable to make that guarantee, the implementation **must** return `VK_ERROR_DEVICE_LOST`. See [Lost Device](#).

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkQueueSubmit` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkQueueSubmit-fence-00063
If `fence` is not `VK_NULL_HANDLE`, `fence` **must** be unsignaled
- VUID-vkQueueSubmit-fence-00064
If `fence` is not `VK_NULL_HANDLE`, `fence` **must** not be associated with any other queue command that has not yet completed execution on that queue

- VUID-vkQueueSubmit-pCommandBuffers-00065
Any calls to [vkCmdSetEvent](#), [vkCmdResetEvent](#) or [vkCmdWaitEvents](#) that have been recorded into any of the command buffer elements of the [pCommandBuffers](#) member of any element of [pSubmits](#), **must** not reference any [VkEvent](#) that is referenced by any of those commands in a command buffer that has been submitted to another queue and is still in the *pending state*
- VUID-vkQueueSubmit-pWaitDstStageMask-00066
Any stage flag included in any element of the [pWaitDstStageMask](#) member of any element of [pSubmits](#) **must** be a pipeline stage supported by one of the capabilities of [queue](#), as specified in the [table of supported pipeline stages](#)
- VUID-vkQueueSubmit-pSignalSemaphores-00067
Each binary semaphore element of the [pSignalSemaphores](#) member of any element of [pSubmits](#) **must** be unsignaled when the semaphore signal operation it defines is executed on the device
- VUID-vkQueueSubmit-pWaitSemaphores-00068
When a semaphore wait operation referring to a binary semaphore defined by any element of the [pWaitSemaphores](#) member of any element of [pSubmits](#) executes on [queue](#), there **must** be no other queues waiting on the same semaphore
- VUID-vkQueueSubmit-pWaitSemaphores-03238
All elements of the [pWaitSemaphores](#) member of all elements of [pSubmits](#) created with a [VkSemaphoreType](#) of [VK_SEMAPHORE_TYPE_BINARY](#) **must** reference a semaphore signal operation that has been submitted for execution and any [semaphore signal operations](#) on which it depends **must** have also been submitted for execution
- VUID-vkQueueSubmit-pCommandBuffers-00070
Each element of the [pCommandBuffers](#) member of each element of [pSubmits](#) **must** be in the *pending or executable state*
- VUID-vkQueueSubmit-pCommandBuffers-00071
If any element of the [pCommandBuffers](#) member of any element of [pSubmits](#) was not recorded with the [VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT](#), it **must** not be in the *pending state*
- VUID-vkQueueSubmit-pCommandBuffers-00072
Any [secondary command buffers recorded](#) into any element of the [pCommandBuffers](#) member of any element of [pSubmits](#) **must** be in the *pending or executable state*
- VUID-vkQueueSubmit-pCommandBuffers-00073
If any [secondary command buffers recorded](#) into any element of the [pCommandBuffers](#) member of any element of [pSubmits](#) was not recorded with the [VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT](#), it **must** not be in the *pending state*
- VUID-vkQueueSubmit-pCommandBuffers-00074
Each element of the [pCommandBuffers](#) member of each element of [pSubmits](#) **must** have been allocated from a [VkCommandPool](#) that was created for the same queue family [queue](#) belongs to
- VUID-vkQueueSubmit-pSubmits-02207
If any element of [pSubmits](#)->[pCommandBuffers](#) includes a [Queue Family Ownership Transfer Acquire Operation](#), there **must** exist a previously submitted [Queue Family Ownership](#)

[Transfer Release Operation](#) on a queue in the queue family identified by the acquire operation, with parameters matching the acquire operation as defined in the definition of such [acquire operations](#), and which happens-before the acquire operation

- VUID-vkQueueSubmit-pSubmits-10911

If any element of `pSubmits->pCommandBuffers` includes a [Queue Family Ownership Transfer Acquire Operation](#), the affected resource **must** not be modified in any way between the last matching release operation and the acquire operation

- VUID-vkQueueSubmit-pCommandBuffers-03220

If a command recorded into any element of `pCommandBuffers` was a `vkCmdBeginQuery` whose `queryPool` was created with a `queryType` of `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, the [profiling lock](#) **must** have been held continuously on the `VkDevice` that `queue` was retrieved from, throughout recording of those command buffers

- VUID-vkQueueSubmit-pSubmits-02808

Any resource created with `VK_SHARING_MODE_EXCLUSIVE` that is read by an operation specified by `pSubmits` **must** not be owned by any queue family other than the one which `queue` belongs to, at the time it is executed

- VUID-vkQueueSubmit-pSubmits-04626

Any resource created with `VK_SHARING_MODE_CONCURRENT` that is accessed by an operation specified by `pSubmits` **must** have included the queue family of `queue` at resource creation time

- VUID-vkQueueSubmit-queue-06448

If `queue` was not created with `VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT`, there **must** be no element of `pSubmits` that includes a `VkProtectedSubmitInfo` structure in its `pNext` chain with `protectedSubmit` equal to `VK_TRUE`

Valid Usage (Implicit)

- VUID-vkQueueSubmit-queue-parameter

`queue` **must** be a valid `VkQueue` handle

- VUID-vkQueueSubmit-pSubmits-parameter

If `submitCount` is not 0, `pSubmits` **must** be a valid pointer to an array of `submitCount` valid `VkSubmitInfo` structures

- VUID-vkQueueSubmit-fence-parameter

If `fence` is not `VK_NULL_HANDLE`, `fence` **must** be a valid `VkFence` handle

- VUID-vkQueueSubmit-commonparent

Both of `fence`, and `queue` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `queue` **must** be externally synchronized
- Host access to `fence` **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
-	-	Any	-

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkSubmitInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSubmitInfo {
    VkStructureType      sType;
    const void*          pNext;
    uint32_t              waitSemaphoreCount;
    const VkSemaphore*    pWaitSemaphores;
    const VkPipelineStageFlags* pWaitDstStageMask;
    uint32_t              commandBufferCount;
    const VkCommandBuffer* pCommandBuffers;
    uint32_t              signalSemaphoreCount;
    const VkSemaphore*    pSignalSemaphores;
} VkSubmitInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `waitSemaphoreCount` is the number of semaphores upon which to wait before executing the command buffers for the batch.
- `pWaitSemaphores` is a pointer to an array of `VkSemaphore` handles upon which to wait before the command buffers for this batch begin execution. If semaphores to wait on are provided, they define a [semaphore wait operation](#).
- `pWaitDstStageMask` is a pointer to an array of pipeline stages at which each corresponding semaphore wait will occur.

- `commandBufferCount` is the number of command buffers to execute in the batch.
- `pCommandBuffers` is a pointer to an array of `VkCommandBuffer` handles to execute in the batch.
- `signalSemaphoreCount` is the number of semaphores to be signaled once the commands specified in `pCommandBuffers` have completed execution.
- `pSignalSemaphores` is a pointer to an array of `VkSemaphore` handles which will be signaled when the command buffers for this batch have completed execution. If semaphores to be signaled are provided, they define a [semaphore signal operation](#).

The order that command buffers appear in `pCommandBuffers` is used to determine [submission order](#), and thus all the [implicit ordering guarantees](#) that respect it. Other than these implicit ordering guarantees and any [explicit synchronization primitives](#), these command buffers **may** overlap or otherwise execute out of order.

Valid Usage

- VUID-VkSubmitInfo-pWaitDstStageMask-04090
If the `geometryShader` feature is not enabled, `pWaitDstStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-VkSubmitInfo-pWaitDstStageMask-04091
If the `tessellationShader` feature is not enabled, `pWaitDstStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-VkSubmitInfo-pWaitDstStageMask-07319
If the `attachmentFragmentShadingRate` feature is not enabled, `pWaitDstStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-VkSubmitInfo-pWaitDstStageMask-03937
If the `synchronization2` feature is not enabled, `pWaitDstStageMask` **must** not be `0`
- VUID-VkSubmitInfo-pCommandBuffers-00075
Each element of `pCommandBuffers` **must** not have been allocated with `VK_COMMAND_BUFFER_LEVEL_SECONDARY`
- VUID-VkSubmitInfo-pWaitDstStageMask-00078
Each element of `pWaitDstStageMask` **must** not include `VK_PIPELINE_STAGE_HOST_BIT`
- VUID-VkSubmitInfo-pWaitSemaphores-03239
If any element of `pWaitSemaphores` or `pSignalSemaphores` was created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`, then the `pNext` chain **must** include a `VkTimelineSemaphoreSubmitInfo` structure
- VUID-VkSubmitInfo-pNext-03240
If the `pNext` chain of this structure includes a `VkTimelineSemaphoreSubmitInfo` structure and any element of `pWaitSemaphores` was created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`, then its `waitSemaphoreValueCount` member **must** equal `waitSemaphoreCount`
- VUID-VkSubmitInfo-pNext-03241
If the `pNext` chain of this structure includes a `VkTimelineSemaphoreSubmitInfo` structure

and any element of `pSignalSemaphores` was created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`, then its `signalSemaphoreValueCount` member **must** equal `signalSemaphoreCount`

- VUID-VkSubmitInfo-pSignalSemaphores-03242

For each element of `pSignalSemaphores` created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE` the corresponding element of `VkTimelineSemaphoreSubmitInfo::pSignalSemaphoreValues` **must** have a value greater than the current value of the semaphore when the semaphore signal operation is executed

- VUID-VkSubmitInfo-pWaitSemaphores-03243

For each element of `pWaitSemaphores` created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE` the corresponding element of `VkTimelineSemaphoreSubmitInfo::pWaitSemaphoreValues` **must** have a value which does not differ from the current value of the semaphore or the value of any outstanding semaphore wait or signal operation on that semaphore by more than `maxTimelineSemaphoreValueDifference`

- VUID-VkSubmitInfo-pSignalSemaphores-03244

For each element of `pSignalSemaphores` created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE` the corresponding element of `VkTimelineSemaphoreSubmitInfo::pSignalSemaphoreValues` **must** have a value which does not differ from the current value of the semaphore or the value of any outstanding semaphore wait or signal operation on that semaphore by more than `maxTimelineSemaphoreValueDifference`

- VUID-VkSubmitInfo-pNext-04120

If the `pNext` chain of this structure does not include a `VkProtectedSubmitInfo` structure with `protectedSubmit` set to `VK_TRUE`, then each element of the `pCommandBuffers` array **must** be an unprotected command buffer

- VUID-VkSubmitInfo-pNext-04148

If the `pNext` chain of this structure includes a `VkProtectedSubmitInfo` structure with `protectedSubmit` set to `VK_TRUE`, then each element of the `pCommandBuffers` array **must** be a protected command buffer

Valid Usage (Implicit)

- VUID-VkSubmitInfo-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_SUBMIT_INFO`

- VUID-VkSubmitInfo-pNext-pNext

Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkDeviceGroupSubmitInfo`, `VkPerformanceQuerySubmitInfoKHR`, `VkProtectedSubmitInfo`, or `VkTimelineSemaphoreSubmitInfo`

- VUID-VkSubmitInfo-sType-unique

The `sType` value of each structure in the `pNext` chain **must** be unique

- VUID-VkSubmitInfo-pWaitSemaphores-parameter
If `waitSemaphoreCount` is not 0, `pWaitSemaphores` **must** be a valid pointer to an array of `waitSemaphoreCount` valid `VkSemaphore` handles
- VUID-VkSubmitInfo-pWaitDstStageMask-parameter
If `waitSemaphoreCount` is not 0, `pWaitDstStageMask` **must** be a valid pointer to an array of `waitSemaphoreCount` valid combinations of `VkPipelineStageFlagBits` values
- VUID-VkSubmitInfo-pCommandBuffers-parameter
If `commandBufferCount` is not 0, `pCommandBuffers` **must** be a valid pointer to an array of `commandBufferCount` valid `VkCommandBuffer` handles
- VUID-VkSubmitInfo-pSignalSemaphores-parameter
If `signalSemaphoreCount` is not 0, `pSignalSemaphores` **must** be a valid pointer to an array of `signalSemaphoreCount` valid `VkSemaphore` handles
- VUID-VkSubmitInfo-commonparent
Each of the elements of `pCommandBuffers`, the elements of `pSignalSemaphores`, and the elements of `pWaitSemaphores` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same `VkDevice`

To specify the values to use when waiting for and signaling semaphores created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`, add a `VkTimelineSemaphoreSubmitInfo` structure to the `pNext` chain of the `VkSubmitInfo` structure when using `vkQueueSubmit`. The `VkTimelineSemaphoreSubmitInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkTimelineSemaphoreSubmitInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           waitSemaphoreValueCount;
    const uint64_t*     pWaitSemaphoreValues;
    uint32_t           signalSemaphoreValueCount;
    const uint64_t*     pSignalSemaphoreValues;
} VkTimelineSemaphoreSubmitInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `waitSemaphoreValueCount` is the number of semaphore wait values specified in `pWaitSemaphoreValues`.
- `pWaitSemaphoreValues` is a pointer to an array of `waitSemaphoreValueCount` values for the corresponding semaphores in `VkSubmitInfo::pWaitSemaphores` to wait for.
- `signalSemaphoreValueCount` is the number of semaphore signal values specified in `pSignalSemaphoreValues`.
- `pSignalSemaphoreValues` is a pointer to an array `signalSemaphoreValueCount` values for the corresponding semaphores in `VkSubmitInfo::pSignalSemaphores` to set when signaled.

If the semaphore in `VkSubmitInfo::pWaitSemaphores` or `VkSubmitInfo::pSignalSemaphores`

corresponding to an entry in `pWaitSemaphoreValues` or `pSignalSemaphoreValues` respectively was not created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`, the implementation **must** ignore the value in the `pWaitSemaphoreValues` or `pSignalSemaphoreValues` entry.

If the semaphore in `VkSubmitInfo::pWaitSemaphores` or `VkSubmitInfo::pSignalSemaphores` corresponding to an entry in `pWaitSemaphoreValues` or `pSignalSemaphoreValues` respectively was created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`, and has `NvSciSyncObj` as the payload, the value in the `pWaitSemaphoreValues` or `pSignalSemaphoreValues` entry **must** be calculated by application via `NvSciSync` APIs.

Valid Usage (Implicit)

- VUID-VkTimelineSemaphoreSubmitInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_TIMELINE_SEMAPHORE_SUBMIT_INFO`
- VUID-VkTimelineSemaphoreSubmitInfo-pWaitSemaphoreValues-parameter
If `waitSemaphoreValueCount` is not 0, and `pWaitSemaphoreValues` is not NULL, `pWaitSemaphoreValues` **must** be a valid pointer to an array of `waitSemaphoreValueCount` `uint64_t` values
- VUID-VkTimelineSemaphoreSubmitInfo-pSignalSemaphoreValues-parameter
If `signalSemaphoreValueCount` is not 0, and `pSignalSemaphoreValues` is not NULL, `pSignalSemaphoreValues` **must** be a valid pointer to an array of `signalSemaphoreValueCount` `uint64_t` values

If the `pNext` chain of `VkSubmitInfo` includes a `VkProtectedSubmitInfo` structure, then the structure indicates whether the batch is protected. The `VkProtectedSubmitInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkProtectedSubmitInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           protectedSubmit;
} VkProtectedSubmitInfo;
```

- `protectedSubmit` specifies whether the batch is protected. If `protectedSubmit` is `VK_TRUE`, the batch is protected. If `protectedSubmit` is `VK_FALSE`, the batch is unprotected. If the `VkSubmitInfo::pNext` chain does not include this structure, the batch is unprotected.

Valid Usage (Implicit)

- VUID-VkProtectedSubmitInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PROTECTED_SUBMIT_INFO`

If the `pNext` chain of `VkSubmitInfo` includes a `VkDeviceGroupSubmitInfo` structure, then that structure includes device indices and masks specifying which physical devices execute semaphore operations and command buffers.

The `VkDeviceGroupSubmitInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDeviceGroupSubmitInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           waitSemaphoreCount;
    const uint32_t*     pWaitSemaphoreDeviceIndices;
    uint32_t           commandBufferCount;
    const uint32_t*     pCommandBufferDeviceMasks;
    uint32_t           signalSemaphoreCount;
    const uint32_t*     pSignalSemaphoreDeviceIndices;
} VkDeviceGroupSubmitInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `waitSemaphoreCount` is the number of elements in the `pWaitSemaphoreDeviceIndices` array.
- `pWaitSemaphoreDeviceIndices` is a pointer to an array of `waitSemaphoreCount` device indices indicating which physical device executes the semaphore wait operation in the corresponding element of `VkSubmitInfo::pWaitSemaphores`.
- `commandBufferCount` is the number of elements in the `pCommandBufferDeviceMasks` array.
- `pCommandBufferDeviceMasks` is a pointer to an array of `commandBufferCount` device masks indicating which physical devices execute the command buffer in the corresponding element of `VkSubmitInfo::pCommandBuffers`. A physical device executes the command buffer if the corresponding bit is set in the mask.
- `signalSemaphoreCount` is the number of elements in the `pSignalSemaphoreDeviceIndices` array.
- `pSignalSemaphoreDeviceIndices` is a pointer to an array of `signalSemaphoreCount` device indices indicating which physical device executes the semaphore signal operation in the corresponding element of `VkSubmitInfo::pSignalSemaphores`.

If this structure is not present, semaphore operations and command buffers execute on device index zero.

Valid Usage

- VUID-VkDeviceGroupSubmitInfo-waitSemaphoreCount-00082
`waitSemaphoreCount` **must** equal `VkSubmitInfo::waitSemaphoreCount`
- VUID-VkDeviceGroupSubmitInfo-commandBufferCount-00083
`commandBufferCount` **must** equal `VkSubmitInfo::commandBufferCount`
- VUID-VkDeviceGroupSubmitInfo-signalSemaphoreCount-00084
`signalSemaphoreCount` **must** equal `VkSubmitInfo::signalSemaphoreCount`
- VUID-VkDeviceGroupSubmitInfo-pWaitSemaphoreDeviceIndices-00085
All elements of `pWaitSemaphoreDeviceIndices` and `pSignalSemaphoreDeviceIndices` **must** be

valid device indices

- VUID-VkDeviceGroupSubmitInfo-pCommandBufferDeviceMasks-00086
All elements of `pCommandBufferDeviceMasks` **must** be valid device masks

Valid Usage (Implicit)

- VUID-VkDeviceGroupSubmitInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_GROUP_SUBMIT_INFO`
- VUID-VkDeviceGroupSubmitInfo-pWaitSemaphoreDeviceIndices-parameter
If `waitSemaphoreCount` is not 0, `pWaitSemaphoreDeviceIndices` **must** be a valid pointer to an array of `waitSemaphoreCount` `uint32_t` values
- VUID-VkDeviceGroupSubmitInfo-pCommandBufferDeviceMasks-parameter
If `commandBufferCount` is not 0, `pCommandBufferDeviceMasks` **must** be a valid pointer to an array of `commandBufferCount` `uint32_t` values
- VUID-VkDeviceGroupSubmitInfo-pSignalSemaphoreDeviceIndices-parameter
If `signalSemaphoreCount` is not 0, `pSignalSemaphoreDeviceIndices` **must** be a valid pointer to an array of `signalSemaphoreCount` `uint32_t` values

If the `pNext` chain of `VkSubmitInfo` includes a `VkPerformanceQuerySubmitInfoKHR` structure, then the structure indicates which counter pass is active for the batch in that submit.

The `VkPerformanceQuerySubmitInfoKHR` structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkPerformanceQuerySubmitInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           counterPassIndex;
} VkPerformanceQuerySubmitInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `counterPassIndex` specifies which counter pass index is active.

If the `VkSubmitInfo::pNext` chain does not include this structure, the batch defaults to use counter pass index 0.

Valid Usage

- VUID-VkPerformanceQuerySubmitInfoKHR-counterPassIndex-03221
`counterPassIndex` **must** be less than the number of counter passes required by any queries within the batch. The required number of counter passes for a performance query is obtained by calling `vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR`

Valid Usage (Implicit)

- `VUID-VkPerformanceQuerySubmitInfoKHR-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PERFORMANCE_QUERY_SUBMIT_INFO_KHR`

6.6. Queue Forward Progress

When using binary semaphores, the application **must** ensure that command buffer submissions will be able to complete without any subsequent operations by the application on any queue. After any call to `vkQueueSubmit` (or other queue operation), for every queued wait on a semaphore created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_BINARY` there **must** be a prior signal of that semaphore that will not be consumed by a different wait on the semaphore.

When using timeline semaphores, wait-before-signal behavior is well-defined and applications **can** submit work via `vkQueueSubmit` defining a [timeline semaphore wait operation](#) before submitting a corresponding [semaphore signal operation](#). For each [timeline semaphore wait operation](#) defined by a call to `vkQueueSubmit`, the application **must** ensure that a corresponding [semaphore signal operation](#) is executed before forward progress can be made.

If a command buffer submission waits for any events to be signaled, the application **must** ensure that command buffer submissions will be able to complete without any subsequent operations by the application. Events signaled by the host **must** be signaled before the command buffer waits on those events.

NOTE

The ability for commands to wait on the host to set an events was originally added to allow low-latency updates to resources between host and device. However, to ensure quality of service, implementations would necessarily detect extended stalls in execution and timeout after a short period. As this period is not defined in the Vulkan specification, it is impossible to correctly validate any application with any wait period. Since the original users of this functionality were highly limited and platform-specific, this functionality is now considered defunct and should not be used.

6.7. Secondary Command Buffer Execution

Secondary command buffers **must** not be directly submitted to a queue. To record a secondary command buffer to execute as part of a primary command buffer, call:

```
// Provided by VK_VERSION_1_0
void vkCmdExecuteCommands(
    VkCommandBuffer          commandBuffer,
    uint32_t                 commandBufferCount,
    const VkCommandBuffer*   pCommandBuffers);
```

- `commandBuffer` is a handle to a primary command buffer that the secondary command buffers

are executed in.

- `commandBufferCount` is the length of the `pCommandBuffers` array.
- `pCommandBuffers` is a pointer to an array of `commandBufferCount` secondary command buffer handles, which are recorded to execute in the primary command buffer in the order they are listed in the array.

If any element of `pCommandBuffers` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` flag, and it was recorded into any other primary command buffer which is currently in the `executable` or `recording` state, that primary command buffer becomes `invalid`.

Valid Usage

- VUID-vkCmdExecuteCommands-pCommandBuffers-00088
Each element of `pCommandBuffers` **must** have been allocated with a `level` of `VK_COMMAND_BUFFER_LEVEL_SECONDARY`
- VUID-vkCmdExecuteCommands-pCommandBuffers-00089
Each element of `pCommandBuffers` **must** be in the `pending` or `executable` state
- VUID-vkCmdExecuteCommands-pCommandBuffers-00091
If any element of `pCommandBuffers` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` flag, it **must** not be in the `pending` state
- VUID-vkCmdExecuteCommands-pCommandBuffers-00092
If any element of `pCommandBuffers` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` flag, it **must** not have already been recorded to `commandBuffer`
- VUID-vkCmdExecuteCommands-pCommandBuffers-00093
If any element of `pCommandBuffers` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT` flag, it **must** not appear more than once in `pCommandBuffers`
- VUID-vkCmdExecuteCommands-pCommandBuffers-00094
Each element of `pCommandBuffers` **must** have been allocated from a `VkCommandPool` that was created for the same queue family as the `VkCommandPool` from which `commandBuffer` was allocated
- VUID-vkCmdExecuteCommands-pCommandBuffers-00096
If this command is called within a render pass instance, each element of `pCommandBuffers` **must** have been recorded with the `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT`
- VUID-vkCmdExecuteCommands-pCommandBuffers-00099
If this command is called within a render pass instance, and any element of `pCommandBuffers` was recorded with `VkCommandBufferInheritanceInfo::framebuffer` not equal to `VK_NULL_HANDLE`, that `VkFramebuffer` **must** match the `VkFramebuffer` used in the current render pass instance
- VUID-vkCmdExecuteCommands-contents-09680
If this command is called within a render pass instance begun with `vkCmdBeginRenderPass`, and `vkCmdNextSubpass` has not been called in the current

render pass instance, the `contents` parameter of `vkCmdBeginRenderPass` **must** have been `VK_SUBPASS_CONTENTS_SECONDARY_COMMAND_BUFFERS`

- VUID-vkCmdExecuteCommands-None-09681

If this command is called within a render pass instance begun with `vkCmdBeginRenderPass`, and `vkCmdNextSubpass` has been called in the current render pass instance, the `contents` parameter of the last call to `vkCmdNextSubpass` **must** have been `VK_SUBPASS_CONTENTS_SECONDARY_COMMAND_BUFFERS`

- VUID-vkCmdExecuteCommands-pCommandBuffers-06019

If this command is called within a render pass instance begun with `vkCmdBeginRenderPass`, each element of `pCommandBuffers` **must** have been recorded with `VkCommandBufferInheritanceInfo::subpass` set to the index of the subpass which the given command buffer will be executed in

- VUID-vkCmdExecuteCommands-pBeginInfo-06020

If this command is called within a render pass instance begun with `vkCmdBeginRenderPass`, the render passes specified in the `pBeginInfo->pInheritanceInfo->renderPass` members of the `vkBeginCommandBuffer` commands used to begin recording each element of `pCommandBuffers` **must** be `compatible` with the current render pass

- VUID-vkCmdExecuteCommands-pCommandBuffers-00100

If `vkCmdExecuteCommands` is not being called within a render pass instance, each element of `pCommandBuffers` **must** not have been recorded with the `VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT`

- VUID-vkCmdExecuteCommands-commandBuffer-00101

If the `inheritedQueries` feature is not enabled, `commandBuffer` **must** not have any queries `active`

- VUID-vkCmdExecuteCommands-commandBuffer-00102

If `commandBuffer` has a `VK_QUERY_TYPE_OCCLUSION` query `active`, then each element of `pCommandBuffers` **must** have been recorded with `VkCommandBufferInheritanceInfo::occlusionQueryEnable` set to `VK_TRUE`

- VUID-vkCmdExecuteCommands-commandBuffer-00103

If `commandBuffer` has a `VK_QUERY_TYPE_OCCLUSION` query `active`, then each element of `pCommandBuffers` **must** have been recorded with `VkCommandBufferInheritanceInfo::queryFlags` having all bits set that are set for the query

- VUID-vkCmdExecuteCommands-commandBuffer-00104

If `commandBuffer` has a `VK_QUERY_TYPE_PIPELINE_STATISTICS` query `active`, then each element of `pCommandBuffers` **must** have been recorded with `VkCommandBufferInheritanceInfo::pipelineStatistics` having all bits set that are set in the `VkQueryPool` the query uses

- VUID-vkCmdExecuteCommands-pCommandBuffers-00105

Each element of `pCommandBuffers` **must** not begin any query types that are `active` in `commandBuffer`

- VUID-vkCmdExecuteCommands-commandBuffer-07594

`commandBuffer` **must** not have any queries other than `VK_QUERY_TYPE_OCCLUSION` and `VK_QUERY_TYPE_PIPELINE_STATISTICS` `active`

- VUID-vkCmdExecuteCommands-commandBuffer-01820

If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, each element of `pCommandBuffers` **must** be a protected command buffer

- VUID-vkCmdExecuteCommands-commandBuffer-01821

If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, each element of `pCommandBuffers` **must** be an unprotected command buffer

- VUID-vkCmdExecuteCommands-commandBuffer-06533

If this command is called within a render pass instance and any recorded command in `commandBuffer` in the current subpass will write to an image subresource as an attachment, commands recorded in elements of `pCommandBuffers` **must** not read from the memory backing that image subresource in any other way

- VUID-vkCmdExecuteCommands-commandBuffer-06534

If this command is called within a render pass instance and any recorded command in `commandBuffer` in the current subpass will read from an image subresource used as an attachment in any way other than as an attachment, commands recorded in elements of `pCommandBuffers` **must** not write to that image subresource as an attachment

- VUID-vkCmdExecuteCommands-pCommandBuffers-06535

If this command is called within a render pass instance and any recorded command in a given element of `pCommandBuffers` will write to an image subresource as an attachment, commands recorded in elements of `pCommandBuffers` at a higher index **must** not read from the memory backing that image subresource in any other way

- VUID-vkCmdExecuteCommands-pCommandBuffers-06536

If this command is called within a render pass instance and any recorded command in a given element of `pCommandBuffers` will read from an image subresource used as an attachment in any way other than as an attachment, commands recorded in elements of `pCommandBuffers` at a higher index **must** not write to that image subresource as an attachment

- VUID-vkCmdExecuteCommands-commandBuffer-09375

`commandBuffer` **must** not be a [secondary command buffer](#)

Valid Usage (Implicit)

- VUID-vkCmdExecuteCommands-commandBuffer-parameter

`commandBuffer` **must** be a valid [VkCommandBuffer](#) handle

- VUID-vkCmdExecuteCommands-pCommandBuffers-parameter

`pCommandBuffers` **must** be a valid pointer to an array of `commandBufferCount` valid [VkCommandBuffer](#) handles

- VUID-vkCmdExecuteCommands-commandBuffer-recording

`commandBuffer` **must** be in the [recording state](#)

- VUID-vkCmdExecuteCommands-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations

- VUID-vkCmdExecuteCommands-commandBufferCount-arraylength

`commandBufferCount` **must** be greater than 0

- VUID-vkCmdExecuteCommands-commonparent

Both of `commandBuffer`, and the elements of `pCommandBuffers` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Indirection

6.8. Command Buffer Device Mask

Each command buffer has a piece of state storing the current device mask of the command buffer. This mask controls which physical devices within the logical device all subsequent commands will execute on, including state-setting commands, action commands, and synchronization commands.

Scissor and viewport state (excluding the count of each) **can** be different values on each physical device (only when set as dynamic state), and each physical device will render using its local copy of the state. Other state is shared between physical devices, such that all physical devices use the most recently set values for the state. However, when recording an action command that uses a piece of state, the most recent command that set that state **must** have included all physical devices that execute the action command in its current device mask.

The command buffer's device mask is orthogonal to the `pCommandBufferDeviceMasks` member of `VkDeviceGroupSubmitInfo`. Commands only execute on a physical device if the device index is set in both device masks.

If the `pNext` chain of `VkCommandBufferBeginInfo` includes a `VkDeviceGroupCommandBufferBeginInfo` structure, then that structure includes an initial device mask for the command buffer.

The `VkDeviceGroupCommandBufferBeginInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDeviceGroupCommandBufferBeginInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           deviceMask;
} VkDeviceGroupCommandBufferBeginInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `deviceMask` is the initial value of the command buffer's device mask.

The initial device mask also acts as an upper bound on the set of devices that **can** ever be in the device mask in the command buffer.

If this structure is not present, the initial value of a command buffer's device mask includes all physical devices in the logical device when the command buffer begins recording.

Valid Usage

- VUID-VkDeviceGroupCommandBufferBeginInfo-deviceMask-00106
`deviceMask` **must** be a valid device mask value
- VUID-VkDeviceGroupCommandBufferBeginInfo-deviceMask-00107
`deviceMask` **must** not be zero

Valid Usage (Implicit)

- VUID-VkDeviceGroupCommandBufferBeginInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_GROUP_COMMAND_BUFFER_BEGIN_INFO`

To update the current device mask of a command buffer, call:

```
// Provided by VK_VERSION_1_1
void vkCmdSetDeviceMask(
    VkCommandBuffer          commandBuffer,
    uint32_t                 deviceMask);
```

- `commandBuffer` is command buffer whose current device mask is modified.
- `deviceMask` is the new value of the current device mask.

`deviceMask` is used to filter out subsequent commands from executing on all physical devices whose bit indices are not set in the mask, except commands beginning a render pass instance, commands transitioning to the next subpass in the render pass instance, and commands ending a render pass instance, which always execute on the set of physical devices whose bit indices are included in the

`deviceMask` member of the `VkDeviceGroupRenderPassBeginInfo` structure passed to the command beginning the corresponding render pass instance.

Valid Usage

- VUID-vkCmdSetDeviceMask-deviceMask-00108
`deviceMask` **must** be a valid device mask value
- VUID-vkCmdSetDeviceMask-deviceMask-00109
`deviceMask` **must** not be zero
- VUID-vkCmdSetDeviceMask-deviceMask-00110
`deviceMask` **must** not include any set bits that were not in the `VkDeviceGroupCommandBufferBeginInfo::deviceMask` value when the command buffer began recording
- VUID-vkCmdSetDeviceMask-deviceMask-00111
If `vkCmdSetDeviceMask` is called inside a render pass instance, `deviceMask` **must** not include any set bits that were not in the `VkDeviceGroupRenderPassBeginInfo::deviceMask` value when the render pass instance began recording

Valid Usage (Implicit)

- VUID-vkCmdSetDeviceMask-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetDeviceMask-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetDeviceMask-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	State

Chapter 7. Synchronization and Cache Control

Synchronization of access to resources is primarily the responsibility of the application in Vulkan. The order of execution of commands with respect to the host and other commands on the device has few implicit guarantees, and needs to be explicitly specified. Memory caches and other optimizations are also explicitly managed, requiring that the flow of data through the system is largely under application control.

Whilst some implicit guarantees exist between commands, five explicit synchronization mechanisms are exposed by Vulkan:

Fences

Fences **can** be used to communicate to the host that execution of some task on the device has completed, controlling resource access between host and device.

Semaphores

Semaphores **can** be used to control resource access across multiple queues.

Events

Events provide a fine-grained synchronization primitive which **can** be signaled either within a command buffer or by the host, and **can** be waited upon within a command buffer or queried on the host. Events **can** be used to control resource access within a single queue.

Pipeline Barriers

Pipeline barriers also provide synchronization control within a command buffer, but at a single point, rather than with separate signal and wait operations. Pipeline barriers **can** be used to control resource access within a single queue.

Render Pass Objects

Render pass objects provide a synchronization framework for rendering tasks, built upon the concepts in this chapter. Many cases that would otherwise need an application to use other synchronization primitives **can** be expressed more efficiently as part of a render pass. Render pass objects **can** be used to control resource access within a single queue.

7.1. Execution and Memory Dependencies

An *operation* is an arbitrary amount of work to be executed on the host, a device, or an external entity such as a presentation engine. Synchronization commands introduce explicit *execution dependencies*, and *memory dependencies* between two sets of operations defined by the command's two *synchronization scopes*.

The synchronization scopes define which other operations a synchronization command is able to create execution dependencies with. Any type of operation that is not in a synchronization command's synchronization scopes will not be included in the resulting dependency. For example, for many synchronization commands, the synchronization scopes **can** be limited to just operations executing in specific [pipeline stages](#), which allows other pipeline stages to be excluded from a

dependency. Other scoping options are possible, depending on the particular command.

An *execution dependency* is a guarantee that for two sets of operations, the first set **must** happen-before the second set. If an operation happens-before another operation, then the first operation **must** complete before the second operation is initiated. More precisely:

- Let **Ops₁** and **Ops₂** be separate sets of operations.
- Let **Sync** be a synchronization command.
- Let **Scope_{1st}** and **Scope_{2nd}** be the synchronization scopes of **Sync**.
- Let **ScopedOps₁** be the intersection of sets **Ops₁** and **Scope_{1st}**.
- Let **ScopedOps₂** be the intersection of sets **Ops₂** and **Scope_{2nd}**.
- Submitting **Ops₁**, **Sync** and **Ops₂** for execution, in that order, will result in execution dependency **ExeDep** between **ScopedOps₁** and **ScopedOps₂**.
- Execution dependency **ExeDep** guarantees that **ScopedOps₁** happen-before **ScopedOps₂**.

An *execution dependency chain* is a sequence of execution dependencies that form a happens-before relation between the first dependency's **ScopedOps₁** and the final dependency's **ScopedOps₂**. For each consecutive pair of execution dependencies, a chain exists if the intersection of **Scope_{2nd}** in the first dependency and **Scope_{1st}** in the second dependency is not an empty set. The formation of a single execution dependency from an execution dependency chain can be described by substituting the following in the description of execution dependencies:

- Let **Sync** be a set of synchronization commands that generate an execution dependency chain.
- Let **Scope_{1st}** be the first synchronization scope of the first command in **Sync**.
- Let **Scope_{2nd}** be the second synchronization scope of the last command in **Sync**.

Execution dependencies alone are not sufficient to guarantee that values resulting from writes in one set of operations **can** be read from another set of operations.

Three additional types of operations are used to control memory access. *Availability operations* cause the values generated by specified memory write accesses to become *available* to a memory domain for future access. Any available value remains available until a subsequent write to the same memory location occurs (whether it is made available or not) or the memory is freed. *Memory domain operations* cause writes that are available to a source memory domain to become available to a destination memory domain (an example of this is making writes available to the host domain available to the device domain). *Visibility operations* cause values available to a memory domain to become *visible* to specified memory accesses.

Availability, visibility, memory domains, and memory domain operations are formally defined in the [Availability and Visibility](#) section of the [Memory Model](#) chapter. Which API operations perform each of these operations is defined in [Availability, Visibility, and Domain Operations](#).

A *memory dependency* is an execution dependency which includes availability and visibility operations such that:

- The first set of operations happens-before the availability operation.

- The availability operation happens-before the visibility operation.
- The visibility operation happens-before the second set of operations.

Once written values are made visible to a particular type of memory access, they **can** be read or written by that type of memory access. Most synchronization commands in Vulkan define a memory dependency.

The specific memory accesses that are made available and visible are defined by the *access scopes* of a memory dependency. Any type of access that is in a memory dependency's first access scope and occurs in **ScopedOps₁** is made available. Any type of access that is in a memory dependency's second access scope and occurs in **ScopedOps₂** has any available writes made visible to it. Any type of operation that is not in a synchronization command's access scopes will not be included in the resulting dependency.

A memory dependency enforces availability and visibility of memory accesses and execution order between two sets of operations. Adding to the description of [execution dependency chains](#):

- Let **MemOps₁** be the set of memory accesses performed by **ScopedOps₁**.
- Let **MemOps₂** be the set of memory accesses performed by **ScopedOps₂**.
- Let **AccessScope_{1st}** be the first access scope of the first command in the **Sync** chain.
- Let **AccessScope_{2nd}** be the second access scope of the last command in the **Sync** chain.
- Let **ScopedMemOps₁** be the intersection of sets **MemOps₁** and **AccessScope_{1st}**.
- Let **ScopedMemOps₂** be the intersection of sets **MemOps₂** and **AccessScope_{2nd}**.
- Submitting **Ops₁**, **Sync**, and **Ops₂** for execution, in that order, will result in a memory dependency **MemDep** between **ScopedOps₁** and **ScopedOps₂**.
- Memory dependency **MemDep** guarantees that:
 - Memory writes in **ScopedMemOps₁** are made available.
 - Available memory writes, including those from **ScopedMemOps₁**, are made visible to **ScopedMemOps₂**.

NOTE

Execution and memory dependencies are used to solve data hazards, i.e. to ensure that read and write operations occur in a well-defined order. Write-after-read hazards can be solved with just an execution dependency, but read-after-write and write-after-write hazards need appropriate memory dependencies to be included between them. If an application does not include dependencies to solve these hazards, the results and execution orders of memory accesses are undefined.

7.1.1. Image Layout Transitions

Image subresources **can** be transitioned from one [layout](#) to another as part of a [memory dependency](#) (e.g. by using an [image memory barrier](#)). When a layout transition is specified in a memory dependency, it happens-after the availability operations in the memory dependency, and happens-before the visibility operations. Image layout transitions **may** perform read and write accesses on all memory bound to the image subresource range, so applications **must** ensure that all memory writes have been made [available](#) before a layout transition is executed. Available memory

is automatically made visible to a layout transition, and writes performed by a layout transition are automatically made available.

Layout transitions always apply to a particular image subresource range, and specify both an old layout and new layout. The old layout **must** either be `VK_IMAGE_LAYOUT_UNDEFINED`, or match the current layout of the image subresource range. If the old layout matches the current layout of the image subresource range, the transition preserves the contents of that range. If the old layout is `VK_IMAGE_LAYOUT_UNDEFINED`, the contents of that range **may** be discarded.

NOTE

Image layout transitions with `VK_IMAGE_LAYOUT_UNDEFINED` allow the implementation to discard the image subresource range, which can provide performance or power benefits. Tile-based architectures may be able to avoid flushing tile data to memory, and immediate style renderers may be able to achieve fast metadata clears to reinitialize frame buffer compression state, or similar.

If the contents of an attachment are not needed after a render pass completes, then applications **should** use `VK_ATTACHMENT_STORE_OP_DONT_CARE`.

As image layout transitions **may** perform read and write accesses on the memory bound to the image, if the image subresource affected by the layout transition is bound to peer memory for any device in the current device mask then the memory heap the bound memory comes from **must** support the `VK_PEER_MEMORY_FEATURE_GENERIC_SRC_BIT` and `VK_PEER_MEMORY_FEATURE_GENERIC_DST_BIT` capabilities as returned by `vkGetDeviceGroupPeerMemoryFeatures`.

NOTE

Applications **must** ensure that layout transitions happen-after all operations accessing the image with the old layout, and happen-before any operations that will access the image with the new layout. Layout transitions are potentially read/write operations, so not defining appropriate memory dependencies to guarantee this will result in a data race.

Image layout transitions interact with [memory aliasing](#).

Layout transitions that are performed via image memory barriers execute in their entirety in [submission order](#), relative to other image layout transitions submitted to the same queue, including those performed by [render passes](#). This ordering of image layout transitions only applies if the implementation performs actual read/write operations during the transition. An application **must** not rely on ordering of image layout transitions to influence ordering of other commands.

The image layout of each image subresource of a depth/stencil image created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` is dependent on the last sample locations used to render to the image subresource as a depth/stencil attachment, thus when the `image` member of an [image memory barrier](#) is an image created with this flag the application **can** chain a `VkSampleLocationsInfoEXT` structure to the `pNext` chain of `VkImageMemoryBarrier2` or `VkImageMemoryBarrier` to specify the sample locations to use during any image layout transition.

If the `VkSampleLocationsInfoEXT` structure does not match the sample location state last used to render to the image subresource range specified by `subresourceRange`, or if no `VkSampleLocationsInfoEXT` structure is present, then the contents of the given image subresource range becomes undefined as if `oldLayout` would equal `VK_IMAGE_LAYOUT_UNDEFINED`.

7.1.2. Pipeline Stages

The work performed by an [action command](#) consists of multiple operations, which are performed as a sequence of logically independent steps known as *pipeline stages*. The exact pipeline stages executed depend on the particular command that is used, and current command buffer state when the command was recorded.

NOTE Operations performed by synchronization commands (e.g. [availability and visibility operations](#)) are not executed by a defined pipeline stage. However other commands can still synchronize with them by using the [synchronization scopes](#) to create a [dependency chain](#).

Execution of operations across pipeline stages **must** adhere to [implicit ordering guarantees](#), particularly including [pipeline stage order](#). Otherwise, execution across pipeline stages **may** overlap or execute out of order with regards to other stages, unless otherwise enforced by an execution dependency.

Several of the synchronization commands include pipeline stage parameters, restricting the [synchronization scopes](#) for that command to just those stages. This allows fine grained control over the exact execution dependencies and accesses performed by action commands. Implementations **should** use these pipeline stages to avoid unnecessary stalls or cache flushing.

Bits which **can** be set in a [VkPipelineStageFlags2](#) mask, specifying stages of execution, are:

```
// Flag bits for VkPipelineStageFlagBits2
typedef VkFlags64 VkPipelineStageFlagBits2;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_NONE = 0ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_TOP_OF_PIPE_BIT =
0x00000001ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT =
0x00000002ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT =
0x00000004ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT =
0x00000008ULL;
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT = 0x00000010ULL;
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT = 0x00000020ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT =
0x00000040ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT =
0x00000080ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT =
0x00000100ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT =
0x00000200ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT
= 0x00000400ULL;
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT =
```

```

0x00000800ULL;
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT =
0x00001000ULL;
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_TRANSFER_BIT =
0x00001000ULL;
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_BOTTOM_OF_PIPE_BIT =
0x00002000ULL;
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_HOST_BIT = 0x00004000ULL;
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT =
0x00008000ULL;
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT =
0x00010000ULL;
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_COPY_BIT = 0x10000000ULL;
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_RESOLVE_BIT =
0x20000000ULL;
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_BLIT_BIT = 0x40000000ULL;
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_CLEAR_BIT = 0x80000000ULL;
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT =
0x100000000ULL;
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT =
0x200000000ULL;
static const VkPipelineStageFlags2
VK_PIPELINE_STAGE_2_PRE_RASTERIZATION_SHADERS_BIT = 0x400000000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_NONE_KHR = 0ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_TOP_OF_PIPE_BIT_KHR =
0x00000001ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT_KHR =
0x00000002ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT_KHR =
0x00000004ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT_KHR =
0x00000008ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlags2
VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT_KHR = 0x00000010ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlags2
VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT_KHR = 0x00000020ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT_KHR =
0x00000040ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT_KHR =
0x00000080ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlags2 VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT_KHR

```

```

= 0x00000100ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT_KHR
= 0x00000200ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT_KHR = 0x00000400ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT_KHR =
0x00000800ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT_KHR =
0x00001000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_TRANSFER_BIT_KHR =
0x00001000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_BOTTOM_OF_PIPE_BIT_KHR =
0x00002000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_HOST_BIT_KHR =
0x00004000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT_KHR =
0x00008000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT_KHR =
0x00010000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_COPY_BIT_KHR =
0x100000000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_RESOLVE_BIT_KHR =
0x200000000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_BLIT_BIT_KHR =
0x400000000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_CLEAR_BIT_KHR =
0x800000000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT_KHR =
0x1000000000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT_KHR = 0x2000000000ULL;
// Provided by VK_KHR_synchronization2
static const VkPipelineStageFlagBits2
VK_PIPELINE_STAGE_2_PRE_RASTERIZATION_SHADERS_BIT_KHR = 0x4000000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_transform_feedback
static const VkPipelineStageFlagBits2 VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT =

```



```

0x01000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_conditional_rendering
static const VkPipelineStageFlagsBits2
VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT = 0x00040000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_device_generated_commands
static const VkPipelineStageFlagsBits2 VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV =
0x00020000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_device_generated_commands
static const VkPipelineStageFlagsBits2 VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_EXT =
0x00020000ULL;
// Provided by VK_KHR_fragment_shading_rate with VK_KHR_synchronization2
static const VkPipelineStageFlagsBits2
VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR = 0x00400000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_shading_rate_image
static const VkPipelineStageFlagsBits2 VK_PIPELINE_STAGE_2_SHADING_RATE_IMAGE_BIT_NV =
0x00400000ULL;
// Provided by VK_KHR_acceleration_structure with VK_KHR_synchronization2
static const VkPipelineStageFlagsBits2
VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR = 0x02000000ULL;
// Provided by VK_KHR_ray_tracing_pipeline with VK_KHR_synchronization2
static const VkPipelineStageFlagsBits2 VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR =
0x00200000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_ray_tracing
static const VkPipelineStageFlagsBits2 VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_NV =
0x00200000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_ray_tracing
static const VkPipelineStageFlagsBits2
VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_NV = 0x02000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_fragment_density_map
static const VkPipelineStageFlagsBits2
VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT = 0x00800000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_mesh_shader
static const VkPipelineStageFlagsBits2 VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_NV =
0x00080000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_mesh_shader
static const VkPipelineStageFlagsBits2 VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_NV =
0x00100000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_mesh_shader
static const VkPipelineStageFlagsBits2 VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT =
0x00080000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_mesh_shader
static const VkPipelineStageFlagsBits2 VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT =
0x00100000ULL;

```

```

// Provided by VK_KHR_synchronization2
typedef VkPipelineStageFlagsBits2 VkPipelineStageFlagsBits2KHR;

```

- **VK_PIPELINE_STAGE_2_NONE** specifies no stages of execution.
- **VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT** specifies the stage of the pipeline where indirect

command parameters are consumed.

- `VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT` specifies the stage of the pipeline where index buffers are consumed.
- `VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT` specifies the stage of the pipeline where vertex buffers are consumed.
- `VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT` is equivalent to the logical OR of:
 - `VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT`
 - `VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT`
- `VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT` specifies the vertex shader stage.
- `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT` specifies the tessellation control shader stage.
- `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT` specifies the tessellation evaluation shader stage.
- `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT` specifies the geometry shader stage.
- `VK_PIPELINE_STAGE_2_PRE_RASTERIZATION_SHADERS_BIT` is equivalent to specifying all supported [pre-rasterization shader stages](#):
 - `VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT`
 - `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT`
 - `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`
 - `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`
- `VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT` specifies the fragment shader stage.
- `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT` specifies the stage of the pipeline where early fragment tests (depth and stencil tests before fragment shading) are performed. This stage also includes [render pass load operations](#) for framebuffer attachments with a depth/stencil format.
- `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT` specifies the stage of the pipeline where late fragment tests (depth and stencil tests after fragment shading) are performed. This stage also includes [render pass store operations](#) for framebuffer attachments with a depth/stencil format.
- `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT` specifies the stage of the pipeline where final color values are output from the pipeline. This stage includes [blending](#), [logic operations](#), render pass [load](#) and [store](#) operations for color attachments, [render pass multisample resolve operations](#), and [vkCmdClearAttachments](#).
- `VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT` specifies the compute shader stage.
- `VK_PIPELINE_STAGE_2_HOST_BIT` specifies a pseudo-stage indicating execution on the host of reads/writes of device memory. This stage is not invoked by any commands recorded in a command buffer.
- `VK_PIPELINE_STAGE_2_COPY_BIT` specifies the execution of all [copy commands](#), including [vkCmdCopyQueryPoolResults](#).
- `VK_PIPELINE_STAGE_2_BLIT_BIT` specifies the execution of [vkCmdBlitImage](#).
- `VK_PIPELINE_STAGE_2_RESOLVE_BIT` specifies the execution of [vkCmdResolveImage](#).

- `VK_PIPELINE_STAGE_2_CLEAR_BIT` specifies the execution of [clear commands](#), with the exception of `vkCmdClearAttachments`.
- `VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT` is equivalent to specifying all of:
 - `VK_PIPELINE_STAGE_2_COPY_BIT`
 - `VK_PIPELINE_STAGE_2_BLIT_BIT`
 - `VK_PIPELINE_STAGE_2_RESOLVE_BIT`
 - `VK_PIPELINE_STAGE_2_CLEAR_BIT`
 - `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR`
- `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT` specifies the execution of all graphics pipeline stages, and is equivalent to the logical OR of:
 - `VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT`
 - `VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT`
 - `VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT`
 - `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT`
 - `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`
 - `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`
 - `VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT`
 - `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`
 - `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT`
 - `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`
 - `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT` specifies all operations performed by all commands supported on the queue it is used with.
- `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR` specifies the stage of the pipeline where the [fragment shading rate attachment](#) is read to determine the fragment shading rate for portions of a rasterized primitive.
- `VK_PIPELINE_STAGE_2_TOP_OF_PIPE_BIT` is equivalent to `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT` with `VkAccessFlags2` set to `0` when specified in the second synchronization scope, but equivalent to `VK_PIPELINE_STAGE_2_NONE` in the first scope.
- `VK_PIPELINE_STAGE_2_BOTTOM_OF_PIPE_BIT` is equivalent to `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT` with `VkAccessFlags2` set to `0` when specified in the first synchronization scope, but equivalent to `VK_PIPELINE_STAGE_2_NONE` in the second scope.

NOTE

The `TOP` and `BOTTOM` pipeline stages are legacy, and applications should prefer `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT` and `VK_PIPELINE_STAGE_2_NONE`.

NOTE

The `VkPipelineStageFlags2` bitmask goes beyond the 31 individual bit flags allowable within a C99 enum, which is how `VkPipelineStageFlagBits` is defined. The first 31 values are common to both, and are interchangeable.

`VkPipelineStageFlags2` is a bitmask type for setting a mask of zero or more `VkPipelineStageFlagBits2` flags:

```
typedef VkFlags64 VkPipelineStageFlags2;
```

```
// Provided by VK_KHR_synchronization2
typedef VkPipelineStageFlags2 VkPipelineStageFlags2KHR;
```

Bits which **can** be set in a `VkPipelineStageFlags` mask, specifying stages of execution, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPipelineStageFlagBits {
    VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT = 0x00000001,
    VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT = 0x00000002,
    VK_PIPELINE_STAGE_VERTEX_INPUT_BIT = 0x00000004,
    VK_PIPELINE_STAGE_VERTEX_SHADER_BIT = 0x00000008,
    VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT = 0x00000010,
    VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT = 0x00000020,
    VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT = 0x00000040,
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT = 0x00000080,
    VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT = 0x00000100,
    VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT = 0x00000200,
    VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT = 0x00000400,
    VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT = 0x00000800,
    VK_PIPELINE_STAGE_TRANSFER_BIT = 0x00001000,
    VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT = 0x00002000,
    VK_PIPELINE_STAGE_HOST_BIT = 0x00004000,
    VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT = 0x00008000,
    VK_PIPELINE_STAGE_ALL_COMMANDS_BIT = 0x00010000,
    VK_PIPELINE_STAGE_NONE = 0,
    // Provided by VK_KHR_fragment_shading_rate
    VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR = 0x00400000,
    // Provided by VK_KHR_synchronization2
    VK_PIPELINE_STAGE_NONE_KHR = VK_PIPELINE_STAGE_NONE,
} VkPipelineStageFlagBits;
```

These values all have the same meaning as the equivalently named values for `VkPipelineStageFlags2`.

- `VK_PIPELINE_STAGE_NONE` specifies no stages of execution.
- `VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT` specifies the stage of the pipeline where `VkDrawIndirect*` / `VkDispatchIndirect*` / `VkTraceRaysIndirect*` data structures are consumed.
- `VK_PIPELINE_STAGE_VERTEX_INPUT_BIT` specifies the stage of the pipeline where vertex and index buffers are consumed.
- `VK_PIPELINE_STAGE_VERTEX_SHADER_BIT` specifies the vertex shader stage.

- `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` specifies the tessellation control shader stage.
- `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT` specifies the tessellation evaluation shader stage.
- `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT` specifies the geometry shader stage.
- `VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT` specifies the fragment shader stage.
- `VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT` specifies the stage of the pipeline where early fragment tests (depth and stencil tests before fragment shading) are performed. This stage also includes [render pass load operations](#) for framebuffer attachments with a depth/stencil format.
- `VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT` specifies the stage of the pipeline where late fragment tests (depth and stencil tests after fragment shading) are performed. This stage also includes [render pass store operations](#) for framebuffer attachments with a depth/stencil format.
- `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT` specifies the stage of the pipeline after blending where the final color values are output from the pipeline. This stage includes [blending](#), [logic operations](#), render pass [load](#) and [store](#) operations for color attachments, [render pass multisample resolve operations](#), and [vkCmdClearAttachments](#).
- `VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT` specifies the execution of a compute shader.
- `VK_PIPELINE_STAGE_TRANSFER_BIT` specifies the following commands:
 - All [copy commands](#), including [vkCmdCopyQueryPoolResults](#)
 - [vkCmdBlitImage2KHR](#) and [vkCmdBlitImage](#)
 - [vkCmdResolveImage2KHR](#) and [vkCmdResolveImage](#)
 - All [clear commands](#), with the exception of [vkCmdClearAttachments](#)
- `VK_PIPELINE_STAGE_HOST_BIT` specifies a pseudo-stage indicating execution on the host of reads/writes of device memory. This stage is not invoked by any commands recorded in a command buffer.
- `VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT` specifies the execution of all graphics pipeline stages, and is equivalent to the logical OR of:
 - `VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT`
 - `VK_PIPELINE_STAGE_VERTEX_INPUT_BIT`
 - `VK_PIPELINE_STAGE_VERTEX_SHADER_BIT`
 - `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT`
 - `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
 - `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
 - `VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT`
 - `VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT`
 - `VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT`
 - `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT`
 - `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`

- `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT` specifies all operations performed by all commands supported on the queue it is used with.
- `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR` specifies the stage of the pipeline where the [fragment shading rate attachment](#) is read to determine the fragment shading rate for portions of a rasterized primitive.
- `VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT` is equivalent to `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT` with `VkAccessFlags` set to `0` when specified in the second synchronization scope, but specifies no stage of execution when specified in the first scope.
- `VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT` is equivalent to `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT` with `VkAccessFlags` set to `0` when specified in the first synchronization scope, but specifies no stage of execution when specified in the second scope.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineStageFlags;
```

`VkPipelineStageFlags` is a bitmask type for setting a mask of zero or more [VkPipelineStageFlagBits](#).

If a synchronization command includes a source stage mask, its first [synchronization scope](#) only includes execution of the pipeline stages specified in that mask and any [logically earlier](#) stages. Its first [access scope](#) only includes memory accesses performed by pipeline stages explicitly specified in the source stage mask.

If a synchronization command includes a destination stage mask, its second [synchronization scope](#) only includes execution of the pipeline stages specified in that mask and any [logically later](#) stages. Its second [access scope](#) only includes memory accesses performed by pipeline stages explicitly specified in the destination stage mask.

NOTE

Note that [access scopes](#) do not interact with the logically earlier or later stages for either scope - only the stages the application specifies are considered part of each access scope.

Certain pipeline stages are only available on queues that support a particular set of operations. The following table lists, for each pipeline stage flag, which queue capability flag **must** be supported by the queue. When multiple flags are enumerated in the second column of the table, it means that the pipeline stage is supported on the queue if it supports any of the listed capability flags. For further details on queue capabilities see [Physical Device Enumeration](#) and [Queues](#).

Table 4. Supported Pipeline Stage Flags

Pipeline stage flag	Required queue capability flag
<code>VK_PIPELINE_STAGE_2_NONE</code>	None required
<code>VK_PIPELINE_STAGE_2_TOP_OF_PIPE_BIT</code>	None required
<code>VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT</code>	<code>VK_QUEUE_GRAPHICS_BIT</code> or <code>VK_QUEUE_COMPUTE_BIT</code>
<code>VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT</code>	<code>VK_QUEUE_GRAPHICS_BIT</code>

Pipeline stage flag	Required queue capability flag
VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT	VK_QUEUE_COMPUTE_BIT
VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT	VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT or VK_QUEUE_TRANSFER_BIT
VK_PIPELINE_STAGE_2_BOTTOM_OF_PIPE_BIT	None required
VK_PIPELINE_STAGE_2_HOST_BIT	None required
VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT	None required
VK_PIPELINE_STAGE_2_COPY_BIT	VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT or VK_QUEUE_TRANSFER_BIT
VK_PIPELINE_STAGE_2_RESOLVE_BIT	VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT or VK_QUEUE_TRANSFER_BIT
VK_PIPELINE_STAGE_2_BLIT_BIT	VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT or VK_QUEUE_TRANSFER_BIT
VK_PIPELINE_STAGE_2_CLEAR_BIT	VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT or VK_QUEUE_TRANSFER_BIT
VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_PRE_RASTERIZATION_SHADERS_BIT	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR	VK_QUEUE_GRAPHICS_BIT
VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR	VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT or VK_QUEUE_TRANSFER_BIT

Pipeline stages that execute as a result of a command logically complete execution in a specific order, such that completion of a logically later pipeline stage **must** not happen-before completion of a logically earlier stage. This means that including any stage in the source stage mask for a particular synchronization command also implies that any logically earlier stages are included in **Scope_{1st}** for that command.

Similarly, initiation of a logically earlier pipeline stage **must** not happen-after initiation of a logically later pipeline stage. Including any given stage in the destination stage mask for a particular synchronization command also implies that any logically later stages are included in **Scope_{2nd}** for that command.

NOTE

Implementations **may** not support synchronization at every pipeline stage for every synchronization operation. If a pipeline stage that an implementation does not support synchronization for appears in a source stage mask, it **may** substitute any logically later stage in its place for the first synchronization scope. If a pipeline stage that an implementation does not support synchronization for appears in a destination stage mask, it **may** substitute any logically earlier stage in its place for the second synchronization scope.

For example, if an implementation is unable to signal an event immediately after vertex shader execution is complete, it **may** instead signal the event after color attachment output has completed.

If an implementation makes such a substitution, it **must** not affect the semantics of execution or memory dependencies or image and buffer memory barriers.

Graphics pipelines are executable on queues supporting **VK_QUEUE_GRAPHICS_BIT**. Stages executed by graphics pipelines **can** only be specified in commands recorded for queues supporting **VK_QUEUE_GRAPHICS_BIT**.

The graphics pipeline executes the following stages, with the logical ordering of the stages matching the order specified here:

- **VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT**
- **VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT**
- **VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT**
- **VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT**
- **VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT**
- **VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT**
- **VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT**
- **VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR**
- **VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT**
- **VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT**
- **VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT**
- **VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT**

For the compute pipeline, the following stages occur in this order:

- **VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT**
- **VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT**

For the transfer pipeline, the following stages occur in this order:

- `VK_PIPELINE_STAGE_2_TRANSFER_BIT`

For host operations, only one pipeline stage occurs, so no order is guaranteed:

- `VK_PIPELINE_STAGE_2_HOST_BIT`

7.1.3. Access Types

Memory in Vulkan **can** be accessed from within shader invocations and via some fixed-function stages of the pipeline. The *access type* is a function of the [descriptor type](#) used, or how a fixed-function stage accesses memory.

Some synchronization commands take sets of access types as parameters to define the [access scopes](#) of a memory dependency. If a synchronization command includes a *source access mask*, its first [access scope](#) only includes accesses via the access types specified in that mask. Similarly, if a synchronization command includes a *destination access mask*, its second [access scope](#) only includes accesses via the access types specified in that mask.

Bits which **can** be set in the `srcAccessMask` and `dstAccessMask` members of `VkMemoryBarrier2KHR`, `VkImageMemoryBarrier2KHR`, and `VkBufferMemoryBarrier2KHR`, specifying access behavior, are:

```
// Flag bits for VkAccessFlagBits2
typedef VkFlags64 VkAccessFlagBits2;
static const VkAccessFlagBits2 VK_ACCESS_2_NONE = 0ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT = 0x00000001ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_INDEX_READ_BIT = 0x00000002ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT = 0x00000004ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_UNIFORM_READ_BIT = 0x00000008ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT = 0x00000010ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_READ_BIT = 0x00000020ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_WRITE_BIT = 0x00000040ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT = 0x00000080ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT = 0x00000100ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT =
0x00000200ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT =
0x00000400ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFER_READ_BIT = 0x00000800ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFER_WRITE_BIT = 0x00001000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_HOST_READ_BIT = 0x00002000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_HOST_WRITE_BIT = 0x00004000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_MEMORY_READ_BIT = 0x00008000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_MEMORY_WRITE_BIT = 0x00010000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_SAMPLED_READ_BIT = 0x10000000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_STORAGE_READ_BIT = 0x20000000ULL;
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT = 0x40000000ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_NONE_KHR = 0ULL;
```

```

// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT_KHR =
0x00000001ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_INDEX_READ_BIT_KHR = 0x00000002ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT_KHR =
0x00000004ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_UNIFORM_READ_BIT_KHR = 0x00000008ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT_KHR =
0x00000010ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_READ_BIT_KHR = 0x00000020ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_WRITE_BIT_KHR = 0x00000040ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT_KHR =
0x00000080ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT_KHR =
0x00000100ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT_KHR =
0x00000200ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT_KHR =
0x00000400ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFER_READ_BIT_KHR = 0x00000800ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFER_WRITE_BIT_KHR = 0x00001000ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_HOST_READ_BIT_KHR = 0x00002000ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_HOST_WRITE_BIT_KHR = 0x00004000ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_MEMORY_READ_BIT_KHR = 0x00008000ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_MEMORY_WRITE_BIT_KHR = 0x00010000ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_SAMPLED_READ_BIT_KHR =
0x10000000ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_STORAGE_READ_BIT_KHR =
0x20000000ULL;
// Provided by VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT_KHR =
0x40000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_transform_feedback

```

```

static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFORM_FEEDBACK_WRITE_BIT_EXT =
0x02000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_transform_feedback
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT =
0x04000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_transform_feedback
static const VkAccessFlagBits2 VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT =
0x08000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_conditional_rendering
static const VkAccessFlagBits2 VK_ACCESS_2_CONDITIONAL_RENDERING_READ_BIT_EXT =
0x00100000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_device_generated_commands
static const VkAccessFlagBits2 VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_NV =
0x00020000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_device_generated_commands
static const VkAccessFlagBits2 VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_NV =
0x00040000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_device_generated_commands
static const VkAccessFlagBits2 VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_EXT =
0x00020000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_device_generated_commands
static const VkAccessFlagBits2 VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_EXT =
0x00040000ULL;
// Provided by VK_KHR_fragment_shading_rate with VK_KHR_synchronization2
static const VkAccessFlagBits2
VK_ACCESS_2_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR = 0x00800000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_shading_rate_image
static const VkAccessFlagBits2 VK_ACCESS_2_SHADING_RATE_IMAGE_READ_BIT_NV =
0x00800000ULL;
// Provided by VK_KHR_acceleration_structure with VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR =
0x00200000ULL;
// Provided by VK_KHR_acceleration_structure with VK_KHR_synchronization2
static const VkAccessFlagBits2 VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_KHR =
0x00400000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_ray_tracing
static const VkAccessFlagBits2 VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_NV =
0x00200000ULL;
// Provided by VK_KHR_synchronization2 with VK_NV_ray_tracing
static const VkAccessFlagBits2 VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_NV =
0x00400000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_fragment_density_map
static const VkAccessFlagBits2 VK_ACCESS_2_FRAGMENT_DENSITY_MAP_READ_BIT_EXT =
0x01000000ULL;
// Provided by VK_KHR_synchronization2 with VK_EXT_blend_operation_advanced
static const VkAccessFlagBits2 VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT =
0x00080000ULL;

```

```
// Provided by VK_KHR_synchronization2
typedef VkAccessFlagBits2 VkAccessFlagBits2KHR;
```

- `VK_ACCESS_2_NONE` specifies no accesses.
- `VK_ACCESS_2_MEMORY_READ_BIT` specifies all read accesses. It is always valid in any access mask, and is treated as equivalent to setting all `READ` access flags that are valid where it is used.
- `VK_ACCESS_2_MEMORY_WRITE_BIT` specifies all write accesses. It is always valid in any access mask, and is treated as equivalent to setting all `WRITE` access flags that are valid where it is used.
- `VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT` specifies read access to command data read from indirect buffers as part of an indirect drawing or dispatch command. Such access occurs in the `VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT` pipeline stage.
- `VK_ACCESS_2_INDEX_READ_BIT` specifies read access to an index buffer as part of an indexed drawing command, bound by `vkCmdBindIndexBuffer`. Such access occurs in the `VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT` pipeline stage.
- `VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT` specifies read access to a vertex buffer as part of a drawing command, bound by `vkCmdBindVertexBuffers`. Such access occurs in the `VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT` pipeline stage.
- `VK_ACCESS_2_UNIFORM_READ_BIT` specifies read access to a `uniform buffer` in any shader pipeline stage.
- `VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT` specifies read access to an `input attachment` within a render pass during fragment shading. Such access occurs in the `VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT` pipeline stage.
- `VK_ACCESS_2_SHADER_SAMPLED_READ_BIT` specifies read access to a `uniform texel buffer` or `sampled image` in any shader pipeline stage.
- `VK_ACCESS_2_SHADER_STORAGE_READ_BIT` specifies read access to a `storage buffer`, `physical storage buffer`, `storage texel buffer`, or `storage image` in any shader pipeline stage.
- `VK_ACCESS_2_SHADER_READ_BIT` is equivalent to the logical OR of:
 - `VK_ACCESS_2_SHADER_SAMPLED_READ_BIT`
 - `VK_ACCESS_2_SHADER_STORAGE_READ_BIT`
- `VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT` specifies write access to a `storage buffer`, `physical storage buffer`, `storage texel buffer`, or `storage image` in any shader pipeline stage.
- `VK_ACCESS_2_SHADER_WRITE_BIT` is equivalent to `VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT`.
- `VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT` specifies read access to a `color attachment`, such as via `blending` (other than `advanced blend operations`), `logic operations` or certain `render pass load operations`. Such access occurs in the `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT` pipeline stage.
- `VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT` specifies write access to a `color attachment` during a `render pass` or via certain render pass `load`, `store`, and `multisample resolve` operations. Such access occurs in the `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT` pipeline stage.
- `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT` specifies read access to a `depth/stencil`

attachment, via [depth or stencil operations](#) or certain [render pass load operations](#). Such access occurs in the `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT` or `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT` pipeline stages.

- `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT` specifies write access to a [depth/stencil attachment](#), via [depth or stencil operations](#) or certain render pass [load](#) and [store](#) operations. Such access occurs in the `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT` or `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT` pipeline stages.
- `VK_ACCESS_2_TRANSFER_READ_BIT` specifies read access to an image or buffer in a [copy](#) operation. Such access occurs in the `VK_PIPELINE_STAGE_2_COPY_BIT`, `VK_PIPELINE_STAGE_2_BLIT_BIT`, or `VK_PIPELINE_STAGE_2_RESOLVE_BIT` pipeline stages.
- `VK_ACCESS_2_TRANSFER_WRITE_BIT` specifies write access to an image or buffer in a [clear](#) or [copy](#) operation. Such access occurs in the `VK_PIPELINE_STAGE_2_COPY_BIT`, `VK_PIPELINE_STAGE_2_BLIT_BIT`, `VK_PIPELINE_STAGE_2_CLEAR_BIT`, or `VK_PIPELINE_STAGE_2_RESOLVE_BIT` pipeline stages.
- `VK_ACCESS_2_HOST_READ_BIT` specifies read access by a host operation. Accesses of this type are not performed through a resource, but directly on memory. Such access occurs in the `VK_PIPELINE_STAGE_2_HOST_BIT` pipeline stage.
- `VK_ACCESS_2_HOST_WRITE_BIT` specifies write access by a host operation. Accesses of this type are not performed through a resource, but directly on memory. Such access occurs in the `VK_PIPELINE_STAGE_2_HOST_BIT` pipeline stage.
- `VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT` specifies read access to [color attachments](#), including [advanced blend operations](#). Such access occurs in the `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT` pipeline stage.
- `VK_ACCESS_2_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR` specifies read access to a fragment shading rate attachment during rasterization. Such access occurs in the `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR` pipeline stage.

NOTE

In situations where an application wishes to select all access types for a given set of pipeline stages, `VK_ACCESS_2_MEMORY_READ_BIT` or `VK_ACCESS_2_MEMORY_WRITE_BIT` can be used. This is particularly useful when specifying stages that only have a single access type.

NOTE

The `VkAccessFlags2` bitmask goes beyond the 31 individual bit flags allowable within a C99 enum, which is how [VkAccessFlagBits](#) is defined. The first 31 values are common to both, and are interchangeable.

`VkAccessFlags2` is a bitmask type for setting a mask of zero or more [VkAccessFlagBits2](#):

```
typedef VkFlags64 VkAccessFlags2;
```

```
// Provided by VK_KHR_synchronization2
typedef VkAccessFlags2 VkAccessFlags2KHR;
```

Bits which **can** be set in the `srcAccessMask` and `dstAccessMask` members of `VkSubpassDependency`, `VkSubpassDependency2`, `VkMemoryBarrier`, `VkBufferMemoryBarrier`, and `VkImageMemoryBarrier`, specifying access behavior, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkAccessFlagBits {
    VK_ACCESS_INDIRECT_COMMAND_READ_BIT = 0x00000001,
    VK_ACCESS_INDEX_READ_BIT = 0x00000002,
    VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT = 0x00000004,
    VK_ACCESS_UNIFORM_READ_BIT = 0x00000008,
    VK_ACCESS_INPUT_ATTACHMENT_READ_BIT = 0x00000010,
    VK_ACCESS_SHADER_READ_BIT = 0x00000020,
    VK_ACCESS_SHADER_WRITE_BIT = 0x00000040,
    VK_ACCESS_COLOR_ATTACHMENT_READ_BIT = 0x00000080,
    VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT = 0x00000100,
    VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT = 0x00000200,
    VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT = 0x00000400,
    VK_ACCESS_TRANSFER_READ_BIT = 0x00000800,
    VK_ACCESS_TRANSFER_WRITE_BIT = 0x00001000,
    VK_ACCESS_HOST_READ_BIT = 0x00002000,
    VK_ACCESS_HOST_WRITE_BIT = 0x00004000,
    VK_ACCESS_MEMORY_READ_BIT = 0x00008000,
    VK_ACCESS_MEMORY_WRITE_BIT = 0x00010000,
    VK_ACCESS_NONE = 0,
    // Provided by VK_EXT_blend_operation_advanced
    VK_ACCESS_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT = 0x00080000,
    // Provided by VK_KHR_fragment_shading_rate
    VK_ACCESS_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR = 0x00800000,
    // Provided by VK_KHR_synchronization2
    VK_ACCESS_NONE_KHR = VK_ACCESS_NONE,
} VkAccessFlagBits;
```

These values all have the same meaning as the equivalently named values for `VkAccessFlags2`.

- `VK_ACCESS_NONE` specifies no accesses.
- `VK_ACCESS_MEMORY_READ_BIT` specifies all read accesses. It is always valid in any access mask, and is treated as equivalent to setting all **READ** access flags that are valid where it is used.
- `VK_ACCESS_MEMORY_WRITE_BIT` specifies all write accesses. It is always valid in any access mask, and is treated as equivalent to setting all **WRITE** access flags that are valid where it is used.
- `VK_ACCESS_INDIRECT_COMMAND_READ_BIT` specifies read access to indirect command data read as part of an indirect drawing or dispatching command. Such access occurs in the `VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT` pipeline stage.
- `VK_ACCESS_INDEX_READ_BIT` specifies read access to an index buffer as part of an indexed drawing command, bound by `vkCmdBindIndexBuffer`. Such access occurs in the `VK_PIPELINE_STAGE_VERTEX_INPUT_BIT` pipeline stage.
- `VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT` specifies read access to a vertex buffer as part of a drawing command, bound by `vkCmdBindVertexBuffers`. Such access occurs in the

VK_PIPELINE_STAGE_VERTEX_INPUT_BIT pipeline stage.

- VK_ACCESS_UNIFORM_READ_BIT specifies read access to a [uniform buffer](#) in any shader pipeline stage.
- VK_ACCESS_INPUT_ATTACHMENT_READ_BIT specifies read access to an [input attachment](#) within a render pass during fragment shading. Such access occurs in the VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT pipeline stage.
- VK_ACCESS_SHADER_READ_BIT specifies read access to a [uniform texel buffer](#), [sampled image](#), [storage buffer](#), [physical storage buffer](#), [storage texel buffer](#), or [storage image](#) in any shader pipeline stage.
- VK_ACCESS_SHADER_WRITE_BIT specifies write access to a [storage buffer](#), [physical storage buffer](#), [storage texel buffer](#), or [storage image](#) in any shader pipeline stage.
- VK_ACCESS_COLOR_ATTACHMENT_READ_BIT specifies read access to a [color attachment](#), such as via [blending](#) (other than [advanced blend operations](#)), [logic operations](#) or certain [render pass load operations](#). Such access occurs in the VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage.
- VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT specifies write access to a [color](#), [resolve](#), or [depth/stencil resolve attachment](#) during a [render pass](#) or via certain render pass [load](#) and [store](#) operations. Such access occurs in the VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage.
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT specifies read access to a [depth/stencil attachment](#), via [depth or stencil operations](#) or certain [render pass load operations](#). Such access occurs in the VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT or VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT pipeline stages.
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT specifies write access to a [depth/stencil attachment](#), via [depth or stencil operations](#) or certain render pass [load](#) and [store](#) operations. Such access occurs in the VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT or VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT pipeline stages.
- VK_ACCESS_TRANSFER_READ_BIT specifies read access to an image or buffer in a [copy](#) operation. Such access occurs in the VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT pipeline stage.
- VK_ACCESS_TRANSFER_WRITE_BIT specifies write access to an image or buffer in a [clear](#) or [copy](#) operation. Such access occurs in the VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT pipeline stage.
- VK_ACCESS_HOST_READ_BIT specifies read access by a host operation. Accesses of this type are not performed through a resource, but directly on memory. Such access occurs in the VK_PIPELINE_STAGE_HOST_BIT pipeline stage.
- VK_ACCESS_HOST_WRITE_BIT specifies write access by a host operation. Accesses of this type are not performed through a resource, but directly on memory. Such access occurs in the VK_PIPELINE_STAGE_HOST_BIT pipeline stage.
- VK_ACCESS_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT specifies read access to [color attachments](#), including [advanced blend operations](#). Such access occurs in the VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT pipeline stage.
- VK_ACCESS_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR specifies read access to a fragment shading rate attachment during rasterization. Such access occurs in the VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR pipeline stage.

Certain access types are only performed by a subset of pipeline stages. Any synchronization command that takes both stage masks and access masks uses both to define the [access scopes](#) - only the specified access types performed by the specified stages are included in the access scope. An application **must** not specify an access flag in a synchronization command if it does not include a pipeline stage in the corresponding stage mask that is able to perform accesses of that type. The following table lists, for each access flag, which pipeline stages **can** perform that type of access.

Table 5. Supported Access Types

Access flag	Supported pipeline stages
VK_ACCESS_2_NONE	Any
VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT	VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT,
VK_ACCESS_2_INDEX_READ_BIT	VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT
VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT	VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT
VK_ACCESS_2_UNIFORM_READ_BIT	VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT,
VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT	VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT,
VK_ACCESS_2_SHADER_READ_BIT	VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT,
VK_ACCESS_2_SHADER_WRITE_BIT	VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT,
VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT	VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT
VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT	VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT

Access flag	Supported pipeline stages
VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT	VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT
VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT	VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT
VK_ACCESS_2_TRANSFER_READ_BIT	VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR,
VK_ACCESS_2_TRANSFER_WRITE_BIT	VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_CLEAR_BIT, VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_COPY_BIT_KHR,
VK_ACCESS_2_HOST_READ_BIT	VK_PIPELINE_STAGE_2_HOST_BIT
VK_ACCESS_2_HOST_WRITE_BIT	VK_PIPELINE_STAGE_2_HOST_BIT
VK_ACCESS_2_MEMORY_READ_BIT	Any
VK_ACCESS_2_MEMORY_WRITE_BIT	Any
VK_ACCESS_2_SHADER_SAMPLED_READ_BIT	VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT,
VK_ACCESS_2_SHADER_STORAGE_READ_BIT	VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT,

Access flag	Supported pipeline stages
VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT	VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT, VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT, VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT, VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT, VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT,
VK_ACCESS_2_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR	VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR
VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT	VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkAccessFlags;
```

VkAccessFlags is a bitmask type for setting a mask of zero or more [VkAccessFlagBits](#).

If a memory object does not have the **VK_MEMORY_PROPERTY_HOST_COHERENT_BIT** property, then [vkFlushMappedMemoryRanges](#) **must** be called in order to guarantee that writes to the memory object from the host are made available to the host domain, where they **can** be further made available to the device domain via a domain operation. Similarly, [vkInvalidateMappedMemoryRanges](#) **must** be called to guarantee that writes which are available to the host domain are made visible to host operations.

If the memory object does have the **VK_MEMORY_PROPERTY_HOST_COHERENT_BIT** property flag, writes to the memory object from the host are automatically made available to the host domain. Similarly, writes made available to the host domain are automatically made visible to the host.

NOTE

[Queue submission commands](#) automatically perform a [domain operation from host to device](#) for all writes performed before the command executes, so in most cases an explicit memory barrier is not needed for this case. In the few circumstances where a submit does not occur between the host write and the device read access, writes **can** be made available by using an explicit memory barrier.

7.1.4. Framebuffer Region Dependencies

[Pipeline stages](#) that operate on, or with respect to, the framebuffer are collectively the *framebuffer-space* pipeline stages. These stages are:

- VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
- VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT

For these pipeline stages, an execution or memory dependency from the first set of operations to the second set **can** either be a single *framebuffer-global* dependency, or split into multiple *framebuffer-local* dependencies. A dependency with non-framebuffer-space pipeline stages is neither framebuffer-global nor framebuffer-local.

A *framebuffer region* is a set of sample (x, y, layer, sample) coordinates that is a subset of the entire framebuffer.

Both [synchronization scopes](#) of a framebuffer-local dependency include only the operations performed within corresponding framebuffer regions (as defined below). No ordering guarantees are made between different framebuffer regions for a framebuffer-local dependency.

Both [synchronization scopes](#) of a framebuffer-global dependency include operations on all framebuffer-regions.

If the first synchronization scope includes operations on pixels/fragments with N samples and the second synchronization scope includes operations on pixels/fragments with M samples, where N does not equal M, then a framebuffer region containing all samples at a given (x, y, layer) coordinate in the first synchronization scope corresponds to a region containing all samples at the same coordinate in the second synchronization scope. In other words, it is a pixel granularity dependency. If N equals M, then a framebuffer region containing a single (x, y, layer, sample) coordinate in the first synchronization scope corresponds to a region containing the same sample at the same coordinate in the second synchronization scope. In other words, it is a sample granularity dependency.

NOTE

Since fragment shader invocations are not specified to run in any particular groupings, the size of a framebuffer region is implementation-dependent, not known to the application, and **must** be assumed to be no larger than specified above.

NOTE

Practically, the pixel vs. sample granularity dependency means that if an input attachment has a different number of samples than the pipeline's [rasterizationSamples](#), then a fragment **can** access any sample in the input attachment's pixel even if it only uses framebuffer-local dependencies. If the input attachment has the same number of samples, then the fragment **can** only access the covered samples in its input [SampleMask](#) (i.e. the fragment operations happen-after a framebuffer-local dependency for each sample the fragment covers). To access samples that are not covered, a framebuffer-global dependency is required.

If a synchronization command includes a [dependencyFlags](#) parameter, and specifies the [VK_DEPENDENCY_BY_REGION_BIT](#) flag, then it defines framebuffer-local dependencies for the framebuffer-space pipeline stages in that synchronization command, for all framebuffer regions. If no [dependencyFlags](#) parameter is included, or the [VK_DEPENDENCY_BY_REGION_BIT](#) flag is not specified, then a framebuffer-global dependency is specified for those stages. The [VK_DEPENDENCY_BY_REGION_BIT](#) flag does not affect the dependencies between non-framebuffer-space pipeline stages, nor does it affect the dependencies between framebuffer-space and non-framebuffer-space pipeline stages.

NOTE

Framebuffer-local dependencies are more efficient for most architectures; particularly tile-based architectures - which can keep framebuffer-regions entirely in on-chip registers and thus avoid external bandwidth across such a dependency. Including a framebuffer-global dependency in your rendering will usually force all implementations to flush data to memory, or to a higher level cache, breaking any potential locality optimizations.

7.1.5. View-Local Dependencies

In a render pass instance that has `multiview` enabled, dependencies **can** be either view-local or view-global.

A view-local dependency only includes operations from a single `source view` from the source subpass in the first synchronization scope, and only includes operations from a single `destination view` from the destination subpass in the second synchronization scope. A view-global dependency includes all views in the view mask of the source and destination subpasses in the corresponding synchronization scopes.

If a synchronization command includes a `dependencyFlags` parameter and specifies the `VK_DEPENDENCY_VIEW_LOCAL_BIT` flag, then it defines view-local dependencies for that synchronization command, for all views. If no `dependencyFlags` parameter is included or the `VK_DEPENDENCY_VIEW_LOCAL_BIT` flag is not specified, then a view-global dependency is specified.

7.1.6. Device-Local Dependencies

Dependencies **can** be either device-local or non-device-local. A device-local dependency acts as multiple separate dependencies, one for each physical device that executes the synchronization command, where each dependency only includes operations from that physical device in both synchronization scopes. A non-device-local dependency is a single dependency where both synchronization scopes include operations from all physical devices that participate in the synchronization command. For subpass dependencies, all physical devices in the `VkDeviceGroupRenderPassBeginInfo::deviceMask` participate in the dependency, and for pipeline barriers all physical devices that are set in the command buffer's current device mask participate in the dependency.

If a synchronization command includes a `dependencyFlags` parameter and specifies the `VK_DEPENDENCY_DEVICE_GROUP_BIT` flag, then it defines a non-device-local dependency for that synchronization command. If no `dependencyFlags` parameter is included or the `VK_DEPENDENCY_DEVICE_GROUP_BIT` flag is not specified, then it defines device-local dependencies for that synchronization command, for all participating physical devices.

Semaphore and event dependencies are device-local and only execute on the one physical device that performs the dependency.

7.2. Implicit Synchronization Guarantees

A small number of implicit ordering guarantees are provided by Vulkan, ensuring that the order in which commands are submitted is meaningful, and avoiding unnecessary complexity in common

operations.

Submission order is a fundamental ordering in Vulkan, giving meaning to the order in which [action and synchronization commands](#) are recorded and submitted to a single queue. Explicit and implicit ordering guarantees between commands in Vulkan all work on the premise that this ordering is meaningful. This order does not itself define any execution or memory dependencies; synchronization commands and other orderings within the API use this ordering to define their scopes.

Submission order for any given set of commands is based on the order in which they were recorded to command buffers and then submitted. This order is determined as follows:

1. The initial order is determined by the order in which [vkQueueSubmit](#) and [vkQueueSubmit2KHR](#) commands are executed on the host, for a single queue, from first to last.
2. The order in which [VkSubmitInfo](#) structures are specified in the [pSubmits](#) parameter of [vkQueueSubmit](#), or in which [VkSubmitInfo2](#) structures are specified in the [pSubmits](#) parameter of [vkQueueSubmit2KHR](#), from lowest index to highest.
3. The order in which command buffers are specified in the [pCommandBuffers](#) member of [VkSubmitInfo](#) or [VkSubmitInfo2](#) from lowest index to highest.
4. The order in which commands outside of a render pass were recorded to a command buffer on the host, from first to last.
5. The order in which commands inside a single subpass were recorded to a command buffer on the host, from first to last.

NOTE

When using a [render pass object](#) with multiple subpasses, commands in different subpasses have no defined submission order relative to each other, regardless of the order in which the subpasses were recorded. Commands within a subpass are still ordered relative to other commands in the same subpass, and those outside of the render pass.

[State commands](#) do not execute any operations on the device, instead they set the state of the command buffer when they execute on the host, in the order that they are recorded. [Action commands](#) consume the current state of the command buffer when they are recorded, and will execute state changes on the device as required to match the recorded state.

[The order of primitives passing through the graphics pipeline](#) and [image layout transitions as part of an image memory barrier](#) provide additional guarantees based on submission order.

Execution of [pipeline stages](#) within a given command also has a loose ordering, dependent only on a single command.

Signal operation order is a fundamental ordering in Vulkan, giving meaning to the order in which semaphore and fence signal operations occur when submitted to a single queue. The signal operation order for queue operations is determined as follows:

1. The initial order is determined by the order in which [vkQueueSubmit](#) and [vkQueueSubmit2KHR](#) commands are executed on the host, for a single queue, from first to last.

2. The order in which `VkSubmitInfo` structures are specified in the `pSubmits` parameter of `vkQueueSubmit`, or in which `VkSubmitInfo2` structures are specified in the `pSubmits` parameter of `vkQueueSubmit2KHR`, from lowest index to highest.
3. The fence signal operation defined by the `fence` parameter of a `vkQueueSubmit` or `vkQueueSubmit2KHR` command is ordered after all semaphore signal operations defined by that command.

Semaphore signal operations defined by a single `VkSubmitInfo` or `VkSubmitInfo2` structure are unordered with respect to other semaphore signal operations defined within the same structure.

The `vkSignalSemaphore` command does not execute on a queue but instead performs the signal operation from the host. The semaphore signal operation defined by executing a `vkSignalSemaphore` command happens-after the `vkSignalSemaphore` command is invoked and happens-before the command returns.

NOTE

When signaling timeline semaphores, it is the responsibility of the application to ensure that they are ordered such that the semaphore value is strictly increasing. Because the first synchronization scope for a semaphore signal operation contains all semaphore signal operations which occur earlier in submission order, all semaphore signal operations contained in any given batch are guaranteed to happen-after all semaphore signal operations contained in any previous batches. However, no ordering guarantee is provided between the semaphore signal operations defined within a single batch. This, combined with the requirement that timeline semaphore values strictly increase, means that it is invalid to signal the same timeline semaphore twice within a single batch.

If an application wishes to ensure that some semaphore signal operation happens-after some other semaphore signal operation, it can submit a separate batch containing only semaphore signal operations, which will happen-after the semaphore signal operations in any earlier batches.

When signaling a semaphore from the host, the only ordering guarantee is that the signal operation happens-after when `vkSignalSemaphore` is called and happens-before it returns. Therefore, it is invalid to call `vkSignalSemaphore` while there are any outstanding signal operations on that semaphore from any queue submissions unless those queue submissions have some dependency which ensures that they happen-after the host signal operation. One example of this would be if the pending signal operation is, itself, waiting on the same semaphore at a lower value and the call to `vkSignalSemaphore` signals that lower value. Furthermore, if there are two or more processes or threads signaling the same timeline semaphore from the host, the application must ensure that the `vkSignalSemaphore` with the lower semaphore value returns before `vkSignalSemaphore` is called with the higher value.

7.3. Fences

Fences are a synchronization primitive that **can** be used to insert a dependency from a queue to the host. Fences have two states - signaled and unsignaled. A fence **can** be signaled as part of the execution of a `queue submission` command. Fences **can** be unsignaled on the host with

[vkResetFences](#). Fences **can** be waited on by the host with the [vkWaitForFences](#) command, and the current state **can** be queried with [vkGetFenceStatus](#).

The internal data of a fence **may** include a reference to any resources and pending work associated with signal or unsignal operations performed on that fence object, collectively referred to as the fence's *payload*. Mechanisms to import and export that internal data to and from fences are provided [below](#). These mechanisms indirectly enable applications to share fence state between two or more fences and other synchronization primitives across process and API boundaries.

Fences are represented by [VkFence](#) handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkFence)
```

To create a fence, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateFence(
    VkDevice                                device,
    const VkFenceCreateInfo*                pCreateInfo,
    const VkAllocationCallbacks*            pAllocator,
    VkFence*                                 pFence);
```

- [device](#) is the logical device that creates the fence.
- [pCreateInfo](#) is a pointer to a [VkFenceCreateInfo](#) structure containing information about how the fence is to be created.
- [pAllocator](#) controls host memory allocation as described in the [Memory Allocation](#) chapter.
- [pFence](#) is a pointer to a handle in which the resulting fence object is returned.

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is [VK_TRUE](#), [vkCreateFence](#) **must** not return [VK_ERROR_OUT_OF_HOST_MEMORY](#).

Valid Usage

- VUID-vkCreateFence-device-05068
The number of fences currently allocated from [device](#) plus 1 **must** be less than or equal to the total number of fences requested via [VkDeviceObjectReservationCreateInfo::fenceRequestCount](#) specified when [device](#) was created
- VUID-vkCreateFence-pNext-05106
If the [pNext](#) chain of [VkFenceCreateInfo](#) includes [VkExportFenceSciSyncInfoNV](#), then [VkFenceCreateInfo::flags](#) **must** not include [VK_FENCE_CREATE_SIGNALED_BIT](#)

Valid Usage (Implicit)

- VUID-vkCreateFence-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkCreateFence-pCreateInfo-parameter
pCreateInfo **must** be a valid pointer to a valid [VkFenceCreateInfo](#) structure
- VUID-vkCreateFence-pAllocator-null
pAllocator **must** be **NULL**
- VUID-vkCreateFence-pFence-parameter
pFence **must** be a valid pointer to a [VkFence](#) handle
- VUID-vkCreateFence-device-queuecount
The device **must** have been created with at least **1** queue

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The [VkFenceCreateInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkFenceCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkFenceCreateFlags  flags;
} VkFenceCreateInfo;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **flags** is a bitmask of [VkFenceCreateFlagBits](#) specifying the initial state and behavior of the fence.

Valid Usage (Implicit)

- VUID-VkFenceCreateInfo-sType-sType

sType must be `VK_STRUCTURE_TYPE_FENCE_CREATE_INFO`

- `VOID-VkFenceCreateInfo-pNext-pNext`

Each **pNext** member of any structure (including this one) in the **pNext** chain must be either `NULL` or a pointer to a valid instance of `VkExportFenceCreateInfo` or `VkExportFenceSciSyncInfoNV`

- `VOID-VkFenceCreateInfo-sType-unique`

The **sType** value of each structure in the **pNext** chain must be unique

- `VOID-VkFenceCreateInfo-flags-parameter`

flags must be a valid combination of `VkFenceCreateFlagBits` values

```
// Provided by VK_VERSION_1_0
typedef enum VkFenceCreateFlagBits {
    VK_FENCE_CREATE_SIGNALED_BIT = 0x00000001,
} VkFenceCreateFlagBits;
```

- `VK_FENCE_CREATE_SIGNALED_BIT` specifies that the fence object is created in the signaled state. Otherwise, it is created in the unsignaled state.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkFenceCreateFlags;
```

`VkFenceCreateFlags` is a bitmask type for setting a mask of zero or more `VkFenceCreateFlagBits`.

To create a fence whose payload **can** be exported to external handles, add a `VkExportFenceCreateInfo` structure to the **pNext** chain of the `VkFenceCreateInfo` structure. The `VkExportFenceCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExportFenceCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkExternalFenceHandleTypeFlags handleTypes;
} VkExportFenceCreateInfo;
```

- **sType** is a `VkStructureType` value identifying this structure.
- **pNext** is `NULL` or a pointer to a structure extending this structure.
- **handleTypes** is a bitmask of `VkExternalFenceHandleTypeFlagBits` specifying one or more fence handle types the application **can** export from the resulting fence. The application **can** request multiple handle types for the same fence.

Valid Usage

- `VOID-VkExportFenceCreateInfo-handleTypes-01446`

The bits in `handleTypes` **must** be supported and compatible, as reported by `VkExternalFenceProperties`

- VUID-VkExportFenceCreateInfo-pNext-05107

If the `pNext` chain includes a `VkExportFenceSciSyncInfoNV` structure, the `VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncFence` and `VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncExport` features, or the `VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncFence` and `VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncExport` features, **must** be enabled

Valid Usage (Implicit)

- VUID-VkExportFenceCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_FENCE_CREATE_INFO`
- VUID-VkExportFenceCreateInfo-handleTypes-parameter
`handleTypes` **must** be a valid combination of `VkExternalFenceHandleTypeFlagBits` values

To export a POSIX file descriptor representing the payload of a fence, call:

```
// Provided by VK_KHR_external_fence_fd
VkResult vkGetFenceFdKHR(
    VkDevice device,
    const VkFenceGetFdInfoKHR* pGetFdInfo,
    int* pFd);
```

- `device` is the logical device that created the fence being exported.
- `pGetFdInfo` is a pointer to a `VkFenceGetFdInfoKHR` structure containing parameters of the export operation.
- `pFd` will return the file descriptor representing the fence payload.

Each call to `vkGetFenceFdKHR` **must** create a new file descriptor and transfer ownership of it to the application. To avoid leaking resources, the application **must** release ownership of the file descriptor when it is no longer needed.

NOTE

Ownership can be released in many ways. For example, the application can call `close()` on the file descriptor, or transfer ownership back to Vulkan by using the file descriptor to import a fence payload.

If `pGetFdInfo->handleType` is `VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT` and the fence is signaled at the time `vkGetFenceFdKHR` is called, `pFd` **may** return the value `-1` instead of a valid file descriptor.

Where supported by the operating system, the implementation **must** set the file descriptor to be closed automatically when an `execve` system call is made.

Exporting a file descriptor from a fence **may** have side effects depending on the transference of the specified handle type, as described in [Importing Fence State](#).

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetFenceFdKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetFenceFdKHR-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetFenceFdKHR-pGetFdInfo-parameter
`pGetFdInfo` **must** be a valid pointer to a valid `VkFenceGetFdInfoKHR` structure
- VUID-vkGetFenceFdKHR-pFd-parameter
`pFd` **must** be a valid pointer to an `int` value

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkFenceGetFdInfoKHR` structure is defined as:

```
// Provided by VK_KHR_external_fence_fd
typedef struct VkFenceGetFdInfoKHR {
    VkStructureType          sType;
    const void*              pNext;
    VkFence                  fence;
    VkExternalFenceHandleTypeFlagBits handleType;
} VkFenceGetFdInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `fence` is the fence from which state will be exported.
- `handleType` is a `VkExternalFenceHandleTypeFlagBits` value specifying the type of handle requested.

The properties of the file descriptor returned depend on the value of `handleType`. See `VkExternalFenceHandleTypeFlagBits` for a description of the properties of the defined external fence handle types.

Valid Usage

- VUID-VkFenceGetFdInfoKHR-handleType-01453
`handleType` **must** have been included in `VkExportFenceCreateInfo::handleTypes` when `fence`'s current payload was created
- VUID-VkFenceGetFdInfoKHR-handleType-01454
If `handleType` refers to a handle type with copy payload transference semantics, `fence` **must** be signaled, or have an associated `fence signal operation` pending execution
- VUID-VkFenceGetFdInfoKHR-fence-01455
`fence` **must** not currently have its payload replaced by an imported payload as described below in `Importing Fence Payloads` unless that imported payload's handle type was included in `VkExternalFenceProperties::exportFromImportedHandleTypes` for `handleType`
- VUID-VkFenceGetFdInfoKHR-handleType-01456
`handleType` **must** be defined as a POSIX file descriptor handle

Valid Usage (Implicit)

- VUID-VkFenceGetFdInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_FENCE_GET_FD_INFO_KHR`
- VUID-VkFenceGetFdInfoKHR-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkFenceGetFdInfoKHR-fence-parameter
`fence` **must** be a valid `VkFence` handle
- VUID-VkFenceGetFdInfoKHR-handleType-parameter
`handleType` **must** be a valid `VkExternalFenceHandleTypeFlagBits` value

To specify additional attributes of `NvSciSync` handles exported from a fence, add a `VkExportFenceSciSyncInfoNV` structure to the `pNext` chain of the `VkFenceCreateInfo` structure. The `VkExportFenceSciSyncInfoNV` structure is defined as:

```
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
typedef struct VkExportFenceSciSyncInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    NvSciSyncAttrList  pAttributes;
} VkExportFenceSciSyncInfoNV;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pAttributes` is an opaque `NvSciSyncAttrList` describing the attributes of the `NvSciSync` object that will be exported.

If [VkExportFenceCreateInfo](#) is not present in the same [pNext](#) chain, this structure is ignored. If the [pNext](#) chain of [VkFenceCreateInfo](#) includes a [VkExportFenceCreateInfo](#) structure with a [NvSciSync handleType](#), but either [VkExportFenceSciSyncInfoNV](#) is not included in the [pNext](#) chain, or it is included but [pAttributes](#) is [NULL](#), [vkCreateFence](#) will return [VK_ERROR_INITIALIZATION_FAILED](#).

The [pAttributes](#) **must** be a reconciled [NvSciSyncAttrList](#). Before exporting the [NvSciSync](#) handles, applications **must** use the [vkGetPhysicalDeviceSciSyncAttributesNV](#) command to get the unreconciled [NvSciSyncAttrList](#) and then use the [NvSciSync](#) API to reconcile it.

Valid Usage

- VUID-VkExportFenceSciSyncInfoNV-pAttributes-05108
[pAttributes](#) **must** be a reconciled [NvSciSyncAttrList](#)

Valid Usage (Implicit)

- VUID-VkExportFenceSciSyncInfoNV-sType-sType
[sType](#) **must** be [VK_STRUCTURE_TYPE_EXPORT_FENCE_SCI_SYNC_INFO_NV](#)

To obtain the implementation-specific [NvSciSync](#) attributes in an unreconciled [NvSciSyncAttrList](#), call:

```
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
VkResult vkGetPhysicalDeviceSciSyncAttributesNV(
    VkPhysicalDevice          physicalDevice,
    const VkSciSyncAttributesInfoNV* pSciSyncAttributesInfo,
    NvSciSyncAttrList          pAttributes);
```

- [physicalDevice](#) is the handle to the physical device that will be used to determine the attributes.
- [pSciSyncAttributesInfo](#) is a pointer to a [VkSciSyncAttributesInfoNV](#) structure containing information about how the attributes are to be filled.
- [pAttributes](#) is an opaque [NvSciSyncAttrList](#) in which the implementation will set the requested attributes.

On success, [pAttributes](#) will contain an unreconciled [NvSciSyncAttrList](#) whose private attributes and some public attributes are filled in by the implementation. If the attributes of [physicalDevice](#) could not be obtained, [VK_ERROR_INITIALIZATION_FAILED](#) is returned.

Valid Usage

- VUID-vkGetPhysicalDeviceSciSyncAttributesNV-pSciSyncAttributesInfo-05109
If [pSciSyncAttributesInfo->primitiveType](#) is [VK_SCI_SYNC_PRIMITIVE_TYPE_FENCE_NV](#) then [VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncFence](#) or [VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncFence](#) **must** be enabled

- VUID-vkGetPhysicalDeviceSciSyncAttributesNV-pSciSyncAttributesInfo-05110
If `pSciSyncAttributesInfo->primitiveType` is `VK_SCI_SYNC_PRIMITIVE_TYPE_SEMAPHORE_NV` then `VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncSemaphore` or `VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncSemaphore2` **must** be enabled
- VUID-vkGetPhysicalDeviceSciSyncAttributesNV-pAttributes-05111
`pAttributes` **must** be a valid `NvSciSyncAttrList` and **must** not be `NULL`

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSciSyncAttributesNV-physicalDevice-parameter
`physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceSciSyncAttributesNV-pSciSyncAttributesInfo-parameter
`pSciSyncAttributesInfo` **must** be a valid pointer to a valid `VkSciSyncAttributesInfoNV` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkSciSyncAttributesInfoNV` structure is defined as:

```
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
typedef struct VkSciSyncAttributesInfoNV {
    VkStructureType      sType;
    const void*          pNext;
    VkSciSyncClientTypeNV clientType;
    VkSciSyncPrimitiveTypeNV primitiveType;
} VkSciSyncAttributesInfoNV;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `clientType` is the permission type of client.
- `primitiveType` is the synchronization primitive type.

`NvSciSync` disallows multi-signalers, therefore clients **must** specify their permission types as one of `signaler`, `waiter` or `signaler_waiter`. In addition, `NvSciSync` requires clients to specify which

primitive type is to be used in synchronization, hence clients also need to provide the primitive type ([VkFence](#) or [VkSemaphore](#)) that will be used.

Valid Usage (Implicit)

- VUID-VkSciSyncAttributesInfoNV-sType-sType
sType must be `VK_STRUCTURE_TYPE_SCI_SYNC_ATTRIBUTES_INFO_NV`
- VUID-VkSciSyncAttributesInfoNV-pNext-pNext
pNext must be `NULL`
- VUID-VkSciSyncAttributesInfoNV-clientType-parameter
clientType must be a valid [VkSciSyncClientTypeNV](#) value
- VUID-VkSciSyncAttributesInfoNV-primitiveType-parameter
primitiveType must be a valid [VkSciSyncPrimitiveTypeNV](#) value

The [VkSciSyncClientTypeNV](#) enum is defined as:

```
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
typedef enum VkSciSyncClientTypeNV {
    VK_SCI_SYNC_CLIENT_TYPE_SIGNALER_NV = 0,
    VK_SCI_SYNC_CLIENT_TYPE_WAITER_NV = 1,
    VK_SCI_SYNC_CLIENT_TYPE_SIGNALER_WAITER_NV = 2,
} VkSciSyncClientTypeNV;
```

- `VK_SCI_SYNC_CLIENT_TYPE_SIGNALER_NV` specifies the permission of the client as signaler. It indicates that the client can only signal the created fence or semaphore and disallows waiting on it.
- `VK_SCI_SYNC_CLIENT_TYPE_WAITER_NV` specifies the permission of the client as waiter. It indicates that the client can only wait on the imported fence or semaphore and disallows signaling it. This type of permission is only used when the client imports `NvSciSync` handles, and export is not allowed.
- `VK_SCI_SYNC_CLIENT_TYPE_SIGNALER_WAITER_NV` specifies the permission of client as both signaler and waiter. It indicates that the client **can** signal and wait on the created fence or semaphore.

The [VkSciSyncPrimitiveTypeNV](#) enum is defined as:

```
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
typedef enum VkSciSyncPrimitiveTypeNV {
    VK_SCI_SYNC_PRIMITIVE_TYPE_FENCE_NV = 0,
    VK_SCI_SYNC_PRIMITIVE_TYPE_SEMAPHORE_NV = 1,
} VkSciSyncPrimitiveTypeNV;
```

- `VK_SCI_SYNC_PRIMITIVE_TYPE_FENCE_NV` specifies that the synchronization primitive type the client will create is a [VkFence](#).
- `VK_SCI_SYNC_PRIMITIVE_TYPE_SEMAPHORE_NV` specifies that the synchronization primitive type the

client will create is a [VkSemaphore](#).

To export a [NvSciSyncFence](#) handle representing the payload of a fence, call:

```
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
VkResult vkGetFenceSciSyncFenceNV(
    VkDevice          device,
    const VkFenceGetSciSyncInfoNV* pGetSciSyncHandleInfo,
    void*             pHandle);
```

- [device](#) is the logical device that created the fence being exported.
- [pGetSciSyncHandleInfo](#) is a pointer to a [VkFenceGetSciSyncInfoNV](#) structure containing parameters of the export operation.
- [pHandle](#) is a pointer to a [NvSciSyncFence](#) which will contain the fence payload on return.

Each call to [vkGetFenceSciSyncFenceNV](#) will duplicate the underlying [NvSciSyncFence](#) handle and transfer the ownership of the [NvSciSyncFence](#) handle to the application. To avoid leaking resources, the application **must** release of the ownership of the [NvSciSyncFence](#) handle when it is no longer needed.

Valid Usage

- VUID-vkGetFenceSciSyncFenceNV-pGetSciSyncHandleInfo-05112
[pGetSciSyncHandleInfo->handleType](#) **must** be
[VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV](#)
- VUID-vkGetFenceSciSyncFenceNV-sciSyncFence-05113
[VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncFence](#) or
[VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncFence](#) **must** be enabled

Valid Usage (Implicit)

- VUID-vkGetFenceSciSyncFenceNV-device-parameter
[device](#) **must** be a valid [VkDevice](#) handle
- VUID-vkGetFenceSciSyncFenceNV-pGetSciSyncHandleInfo-parameter
[pGetSciSyncHandleInfo](#) **must** be a valid pointer to a valid [VkFenceGetSciSyncInfoNV](#) structure
- VUID-vkGetFenceSciSyncFenceNV-pHandle-parameter
[pHandle](#) **must** be a pointer value

Return Codes

Success

- [VK_SUCCESS](#)

Failure

- `VK_ERROR_INVALID_EXTERNAL_HANDLE`
- `VK_ERROR_NOT_PERMITTED`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To export a `NvSciSyncObj` handle representing the payload of a fence, call:

```
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
VkResult vkGetFenceSciSyncObjNV(
    VkDevice          device,
    const VkFenceGetSciSyncInfoNV* pGetSciSyncHandleInfo,
    void*             pHandle);
```

- `device` is the logical device that created the fence being exported.
- `pGetSciSyncHandleInfo` is a pointer to a `VkFenceGetSciSyncInfoNV` structure containing parameters of the export operation.
- `pHandle` will return the `NvSciSyncObj` handle representing the fence payload.

Each call to `vkGetFenceSciSyncObjNV` will duplicate the underlying `NvSciSyncObj` handle and transfer the ownership of the `NvSciSyncObj` handle to the application. To avoid leaking resources, the application **must** release of the ownership of the `NvSciSyncObj` handle when it is no longer needed.

Valid Usage

- VUID-vkGetFenceSciSyncObjNV-pGetSciSyncHandleInfo-05114
`pGetSciSyncHandleInfo->handleType` **must** be
`VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV`
- VUID-vkGetFenceSciSyncObjNV-sciSyncFence-05115
`VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncFence` **or**
`VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncFence` **must** be enabled

Valid Usage (Implicit)

- VUID-vkGetFenceSciSyncObjNV-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetFenceSciSyncObjNV-pGetSciSyncHandleInfo-parameter
`pGetSciSyncHandleInfo` **must** be a valid pointer to a valid `VkFenceGetSciSyncInfoNV` structure
- VUID-vkGetFenceSciSyncObjNV-pHandle-parameter
`pHandle` **must** be a pointer value

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INVALID_EXTERNAL_HANDLE`
- `VK_ERROR_NOT_PERMITTED`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkFenceGetSciSyncInfoNV` structure is defined as:

```
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
typedef struct VkFenceGetSciSyncInfoNV {
    VkStructureType          sType;
    const void*              pNext;
    VkFence                  fence;
    VkExternalFenceHandleTypeFlagBits handleType;
} VkFenceGetSciSyncInfoNV;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `fence` is the fence from which state will be exported.
- `handleType` is the type of `NvSciSync` handle (`NvSciSyncObj` or `NvSciSyncFence`) representing the fence payload that will be exported.

If `handleType` is `VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV`, a `NvSciSyncObj` will be exported. If `handleType` is `VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV`, a `NvSciSyncFence` will be exported.

Valid Usage (Implicit)

- VUID-VkFenceGetSciSyncInfoNV-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_FENCE_GET_SCI_SYNC_INFO_NV`
- VUID-VkFenceGetSciSyncInfoNV-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkFenceGetSciSyncInfoNV-fence-parameter
`fence` **must** be a valid `VkFence` handle
- VUID-VkFenceGetSciSyncInfoNV-handleType-parameter
`handleType` **must** be a valid `VkExternalFenceHandleTypeFlagBits` value

To destroy a fence, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyFence(
    VkDevice          device,
    VkFence           fence,
    const VkAllocationCallbacks* pAllocator);
```

- **device** is the logical device that destroys the fence.
- **fence** is the handle of the fence to destroy.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage

- VUID-vkDestroyFence-fence-01120
All [queue submission](#) commands that refer to **fence** **must** have completed execution

Valid Usage (Implicit)

- VUID-vkDestroyFence-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkDestroyFence-fence-parameter
If **fence** is not [VK_NULL_HANDLE](#), **fence** **must** be a valid [VkFence](#) handle
- VUID-vkDestroyFence-pAllocator-null
pAllocator **must** be [NULL](#)
- VUID-vkDestroyFence-fence-parent
If **fence** is a valid handle, it **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to **fence** **must** be externally synchronized

To query the status of a fence from the host, call:

```
// Provided by VK_VERSION_1_0
VkResult vkGetFenceStatus(
    VkDevice          device,
    VkFence           fence);
```

- **device** is the logical device that owns the fence.
- **fence** is the handle of the fence to query.

Upon success, `vkGetFenceStatus` returns the status of the fence object, with the following return codes:

Table 6. Fence Object Status Codes

Status	Meaning
VK_SUCCESS	The fence specified by <code>fence</code> is signaled.
VK_NOT_READY	The fence specified by <code>fence</code> is unsignaled.
VK_ERROR_DEVICE_LOST	The device has been lost. See Lost Device .

If a [queue submission](#) command is pending execution, then the value returned by this command **may** immediately be out of date.

If the device has been lost (see [Lost Device](#)), `vkGetFenceStatus` **may** return any of the above status codes. If the device has been lost and `vkGetFenceStatus` is called repeatedly, it will eventually return either `VK_SUCCESS` or `VK_ERROR_DEVICE_LOST`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetFenceStatus` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetFenceStatus-device-parameter
`device` **must** be a valid [VkDevice](#) handle
- VUID-vkGetFenceStatus-fence-parameter
`fence` **must** be a valid [VkFence](#) handle
- VUID-vkGetFenceStatus-fence-parent
`fence` **must** have been created, allocated, or retrieved from `device`

Return Codes

Success

- `VK_NOT_READY`
- `VK_SUCCESS`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To set the state of fences to unsignaled from the host, call:

```
// Provided by VK_VERSION_1_0
VkResult vkResetFences(
    VkDevice          device,
    uint32_t          fenceCount,
    const VkFence*    pFences);
```

- **device** is the logical device that owns the fences.
- **fenceCount** is the number of fences to reset.
- **pFences** is a pointer to an array of fence handles to reset.

If any member of **pFences** currently has its **payload imported** with temporary permanence, that fence's prior permanent payload is first restored. The remaining operations described therefore operate on the restored payload.

When **vkResetFences** is executed on the host, it defines a *fence unsignal operation* for each fence, which resets the fence to the unsignaled state.

If any member of **pFences** is already in the unsignaled state when **vkResetFences** is executed, then **vkResetFences** has no effect on that fence.

Valid Usage

- VUID-vkResetFences-pFences-01123
Each element of **pFences** **must** not be currently associated with any queue command that has not yet completed execution on that queue

Valid Usage (Implicit)

- VUID-vkResetFences-device-parameter
device **must** be a valid **VkDevice** handle
- VUID-vkResetFences-pFences-parameter
pFences **must** be a valid pointer to an array of **fenceCount** valid **VkFence** handles
- VUID-vkResetFences-fenceCount-arraylength
fenceCount **must** be greater than 0
- VUID-vkResetFences-pFences-parent
Each element of **pFences** **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to each member of **pFences** **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

When a fence is submitted to a queue as part of a [queue submission](#) command, it defines a memory dependency on the batches that were submitted as part of that command, and defines a *fence signal operation* which sets the fence to the signaled state.

The first [synchronization scope](#) includes every batch submitted in the same [queue submission](#) command. Fence signal operations that are defined by `vkQueueSubmit` or `vkQueueSubmit2KHR` additionally include in the first synchronization scope all commands that occur earlier in [submission order](#). Fence signal operations that are defined by `vkQueueSubmit` or `vkQueueSubmit2KHR` additionally include in the first synchronization scope any semaphore and fence signal operations that occur earlier in [signal operation order](#).

The second [synchronization scope](#) only includes the fence signal operation.

The first [access scope](#) includes all memory access performed by the device.

The second [access scope](#) is empty.

To wait for one or more fences to enter the signaled state on the host, call:

```
// Provided by VK_VERSION_1_0
VkResult vkWaitForFences(
    VkDevice          device,
    uint32_t          fenceCount,
    const VkFence*     pFences,
    VkBool32          waitAll,
    uint64_t           timeout);
```

- `device` is the logical device that owns the fences.
- `fenceCount` is the number of fences to wait on.
- `pFences` is a pointer to an array of `fenceCount` fence handles.
- `waitAll` is the condition that **must** be satisfied to successfully unblock the wait. If `waitAll` is `VK_TRUE`, then the condition is that all fences in `pFences` are signaled. Otherwise, the condition is that at least one fence in `pFences` is signaled.
- `timeout` is the timeout period in units of nanoseconds. `timeout` is adjusted to the closest value allowed by the implementation-dependent timeout accuracy, which **may** be substantially longer

than one nanosecond, and **may** be longer than the requested period.

If the condition is satisfied when `vkWaitForFences` is called, then `vkWaitForFences` returns immediately. If the condition is not satisfied at the time `vkWaitForFences` is called, then `vkWaitForFences` will block and wait until the condition is satisfied or the `timeout` has expired, whichever is sooner.

If `timeout` is zero, then `vkWaitForFences` does not wait, but simply returns the current state of the fences. `VK_TIMEOUT` will be returned in this case if the condition is not satisfied, even though no actual wait was performed.

If the condition is satisfied before the `timeout` has expired, `vkWaitForFences` returns `VK_SUCCESS`. Otherwise, `vkWaitForFences` returns `VK_TIMEOUT` after the `timeout` has expired.

If device loss occurs (see [Lost Device](#)) before the timeout has expired, `vkWaitForFences` **must** return in finite time with either `VK_SUCCESS` or `VK_ERROR_DEVICE_LOST`.

NOTE

While we guarantee that `vkWaitForFences` **must** return in finite time, no guarantees are made that it returns immediately upon device loss. However, the application can reasonably expect that the delay will be on the order of seconds and that calling `vkWaitForFences` will not result in a permanently (or seemingly permanently) dead process.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkWaitForFences` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkWaitForFences-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkWaitForFences-pFences-parameter
`pFences` **must** be a valid pointer to an array of `fenceCount` valid `VkFence` handles
- VUID-vkWaitForFences-fenceCount-arraylength
`fenceCount` **must** be greater than 0
- VUID-vkWaitForFences-pFences-parent
Each element of `pFences` **must** have been created, allocated, or retrieved from `device`

Return Codes

Success

- `VK_SUCCESS`
- `VK_TIMEOUT`

Failure

- `VK_ERROR_DEVICE_LOST`

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

An execution dependency is defined by waiting for a fence to become signaled, either via `vkWaitForFences` or by polling on `vkGetFenceStatus`.

The first [synchronization scope](#) includes only the fence signal operation.

The second [synchronization scope](#) includes the host operations of `vkWaitForFences` or `vkGetFenceStatus` indicating that the fence has become signaled.

NOTE

Signaling a fence and waiting on the host does not guarantee that the results of memory accesses will be visible to the host, as the access scope of a memory dependency defined by a fence only includes device access. A [memory barrier](#) or other memory dependency **must** be used to guarantee this. See the description of [host access types](#) for more information.

7.3.1. Alternate Methods to Signal Fences

Besides submitting a fence to a queue as part of a [queue submission](#) command, a fence **may** also be signaled when a particular event occurs on a device or display.

To create a fence that will be signaled when an event occurs on a device, call:

```
// Provided by VK_EXT_display_control
VkResult vkRegisterDeviceEventEXT(
    VkDevice                                device,
    const VkDeviceEventInfoEXT*             pDeviceInfo,
    const VkAllocationCallbacks*            pAllocator,
    VkFence*                                pFence);
```

- `device` is a logical device on which the event **may** occur.
- `pDeviceInfo` is a pointer to a [VkDeviceEventInfoEXT](#) structure describing the event of interest to the application.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pFence` is a pointer to a handle in which the resulting fence object is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkRegisterDeviceEventEXT` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkRegisterDeviceEventEXT-device-parameter

device must be a valid [VkDevice](#) handle

- VUID-vkRegisterDeviceEventEXT-pDeviceEventInfo-parameter
pDeviceEventInfo must be a valid pointer to a valid [VkDeviceEventInfoEXT](#) structure
- VUID-vkRegisterDeviceEventEXT-pAllocator-null
pAllocator must be **NULL**
- VUID-vkRegisterDeviceEventEXT-pFence-parameter
pFence must be a valid pointer to a [VkFence](#) handle

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The [VkDeviceEventInfoEXT](#) structure is defined as:

```
// Provided by VK_EXT_display_control
typedef struct VkDeviceEventInfoEXT {
    VkStructureType      sType;
    const void*          pNext;
    VkDeviceEventTypeEXT deviceEvent;
} VkDeviceEventInfoEXT;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **deviceEvent** is a [VkDeviceEventTypeEXT](#) value specifying when the fence will be signaled.

Valid Usage (Implicit)

- VUID-VkDeviceEventInfoEXT-sType-sType
sType must be **VK_STRUCTURE_TYPE_DEVICE_EVENT_INFO_EXT**
- VUID-VkDeviceEventInfoEXT-pNext-pNext
pNext must be **NULL**
- VUID-VkDeviceEventInfoEXT-deviceEvent-parameter
deviceEvent must be a valid [VkDeviceEventTypeEXT](#) value

Possible values of [VkDeviceEventInfoEXT::deviceEvent](#), specifying when a fence will be signaled,

are:

```
// Provided by VK_EXT_display_control
typedef enum VkDeviceEventTypeEXT {
    VK_DEVICE_EVENT_TYPE_DISPLAY_HOTPLUG_EXT = 0,
} VkDeviceEventTypeEXT;
```

- `VK_DEVICE_EVENT_TYPE_DISPLAY_HOTPLUG_EXT` specifies that the fence is signaled when a display is plugged into or unplugged from the specified device. Applications **can** use this notification to determine when they need to re-enumerate the available displays on a device.

To create a fence that will be signaled when an event occurs on a `VkDisplayKHR` object, call:

```
// Provided by VK_EXT_display_control
VkResult vkRegisterDisplayEventEXT(
    VkDevice          device,
    VkDisplayKHR      display,
    const VkDisplayEventInfoEXT* pDisplayEventInfo,
    const VkAllocationCallbacks* pAllocator,
    VkFence*          pFence);
```

- `device` is a logical device associated with `display`
- `display` is the display on which the event **may** occur.
- `pDisplayEventInfo` is a pointer to a `VkDisplayEventInfoEXT` structure describing the event of interest to the application.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pFence` is a pointer to a handle in which the resulting fence object is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkRegisterDisplayEventEXT` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkRegisterDisplayEventEXT-device-parameter `device` **must** be a valid `VkDevice` handle
- VUID-vkRegisterDisplayEventEXT-display-parameter `display` **must** be a valid `VkDisplayKHR` handle
- VUID-vkRegisterDisplayEventEXT-pDisplayEventInfo-parameter `pDisplayEventInfo` **must** be a valid pointer to a valid `VkDisplayEventInfoEXT` structure
- VUID-vkRegisterDisplayEventEXT-pAllocator-null `pAllocator` **must** be `NULL`
- VUID-vkRegisterDisplayEventEXT-pFence-parameter `pFence` **must** be a valid pointer to a `VkFence` handle

- VUID-vkRegisterDisplayEventEXT-commonparent
Both of **device**, and **display** **must** have been created, allocated, or retrieved from the same **VkPhysicalDevice**

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The **VkDisplayEventInfoEXT** structure is defined as:

```
// Provided by VK_EXT_display_control
typedef struct VkDisplayEventInfoEXT {
    VkStructureType      sType;
    const void*          pNext;
    VkDisplayEventTypeEXT displayEvent;
} VkDisplayEventInfoEXT;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **displayEvent** is a **VkDisplayEventTypeEXT** specifying when the fence will be signaled.

Valid Usage (Implicit)

- VUID-VkDisplayEventInfoEXT-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_DISPLAY_EVENT_INFO_EXT**
- VUID-VkDisplayEventInfoEXT-pNext-pNext
pNext **must** be **NULL**
- VUID-VkDisplayEventInfoEXT-displayEvent-parameter
displayEvent **must** be a valid **VkDisplayEventTypeEXT** value

Possible values of **VkDisplayEventInfoEXT::displayEvent**, specifying when a fence will be signaled, are:


```
// Provided by VK_EXT_display_control
typedef enum VkDisplayEventTypeEXT {
    VK_DISPLAY_EVENT_TYPE_FIRST_PIXEL_OUT_EXT = 0,
} VkDisplayEventTypeEXT;
```

- `VK_DISPLAY_EVENT_TYPE_FIRST_PIXEL_OUT_EXT` specifies that the fence is signaled when the first pixel of the next display refresh cycle leaves the display engine for the display.

7.3.2. Importing Fence Payloads

Applications **can** import a fence payload into an existing fence using an external fence handle. The effects of the import operation will be either temporary or permanent, as specified by the application. If the import is temporary, the fence will be *restored* to its permanent state the next time that fence is passed to [vkResetFences](#).

NOTE

Restoring a fence to its prior permanent payload is a distinct operation from resetting a fence payload. See [vkResetFences](#) for more detail.

Performing a subsequent temporary import on a fence before resetting it has no effect on this requirement; the next unsignal of the fence **must** still restore its last permanent state. A permanent payload import behaves as if the target fence was destroyed, and a new fence was created with the same handle but the imported payload. Because importing a fence payload temporarily or permanently detaches the existing payload from a fence, similar usage restrictions to those applied to `vkDestroyFence` are applied to any command that imports a fence payload. Which of these import types is used is referred to as the import operation's *permanence*. Each handle type supports either one or both types of permanence.

The implementation **must** perform the import operation by either referencing or copying the payload referred to by the specified external fence handle, depending on the handle's type. The import method used is referred to as the handle type's *transference*. When using handle types with reference transference, importing a payload to a fence adds the fence to the set of all fences sharing that payload. This set includes the fence from which the payload was exported. Fence signaling, waiting, and resetting operations performed on any fence in the set **must** behave as if the set were a single fence. Importing a payload using handle types with copy transference creates a duplicate copy of the payload at the time of import, but makes no further reference to it. Fence signaling, waiting, and resetting operations performed on the target of copy imports **must** not affect any other fence or payload.

Export operations have the same transference as the specified handle type's import operations. Additionally, exporting a fence payload to a handle with copy transference has the same side effects on the source fence's payload as executing a fence reset operation. If the fence was using a temporarily imported payload, the fence's prior permanent payload will be restored.

NOTE

The table [Handle Types Supported by VkImportFenceFdInfoKHR](#) defines the permanence and transference of each handle type.

[External synchronization](#) allows implementations to modify an object's internal state, i.e. payload,

without internal synchronization. However, for fences sharing a payload across processes, satisfying the external synchronization requirements of `VkFence` parameters as if all fences in the set were the same object is sometimes infeasible. Satisfying valid usage constraints on the state of a fence would similarly require impractical coordination or levels of trust between processes. Therefore, these constraints only apply to a specific fence handle, not to its payload. For distinct fence objects which share a payload:

- If multiple commands which queue a signal operation, or which unsignal a fence, are called concurrently, behavior will be as if the commands were called in an arbitrary sequential order.
- If a queue submission command is called with a fence that is sharing a payload, and the payload is already associated with another queue command that has not yet completed execution, either one or both of the commands will cause the fence to become signaled when they complete execution.
- If a fence payload is reset while it is associated with a queue command that has not yet completed execution, the payload will become unsignaled, but **may** become signaled again when the command completes execution.
- In the preceding cases, any of the devices associated with the fences sharing the payload **may** be lost, or any of the queue submission or fence reset commands **may** return `VK_ERROR_INITIALIZATION_FAILED`.

Other than these non-deterministic results, behavior is well defined. In particular:

- The implementation **must** not crash or enter an internally inconsistent state where future valid Vulkan commands might cause undefined results,
- Timeouts on future wait commands on fences sharing the payload **must** be effective.

NOTE

These rules allow processes to synchronize access to shared memory without trusting each other. However, such processes must still be cautious not to use the shared fence for more than synchronizing access to the shared memory. For example, a process should not use a fence with shared payload to tell when commands it submitted to a queue have completed and objects used by those commands may be destroyed, since the other process can accidentally or maliciously cause the fence to signal before the commands actually complete.

When a fence is using an imported payload, its `VkExportFenceCreateInfo::handleTypes` value is specified when creating the fence from which the payload was exported, rather than specified when creating the fence. Additionally, `VkExternalFenceProperties::exportFromImportedHandleTypes` restricts which handle types **can** be exported from such a fence based on the specific handle type used to import the current payload. Passing a fence to `vkAcquireNextImageKHR` is equivalent to temporarily importing a fence payload to that fence.

NOTE

Because the exportable handle types of an imported fence correspond to its current imported payload, and `vkAcquireNextImageKHR` behaves the same as a temporary import operation for which the source fence is opaque to the application, applications have no way of determining whether any external handle types **can** be exported from a fence in this state. Therefore, applications **must** not attempt to export handles from fences using a temporarily imported payload from

When importing a fence payload, it is the responsibility of the application to ensure the external handles meet all valid usage requirements. However, implementations **must** perform sufficient validation of external handles to ensure that the operation results in a valid fence which will not cause program termination, device loss, queue stalls, host thread stalls, or corruption of other resources when used as allowed according to its import parameters. If the external handle provided does not meet these requirements, the implementation **must** fail the fence payload import operation with the error code `VK_ERROR_INVALID_EXTERNAL_HANDLE`.

To import a fence payload from a POSIX file descriptor, call:

```
// Provided by VK_KHR_external_fence_fd
VkResult vkImportFenceFdKHR(
    VkDevice device,
    const VkImportFenceFdInfoKHR* pImportFenceFdInfo);
```

- `device` is the logical device that created the fence.
- `pImportFenceFdInfo` is a pointer to a `VkImportFenceFdInfoKHR` structure specifying the fence and import parameters.

Importing a fence payload from a file descriptor transfers ownership of the file descriptor from the application to the Vulkan implementation. The application **must** not perform any operations on the file descriptor after a successful import.

Applications **can** import the same fence payload into multiple instances of Vulkan, into the same instance from which it was exported, and multiple times into a given Vulkan instance.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkImportFenceFdKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkImportFenceFdKHR-fence-01463
`fence` **must** not be associated with any queue command that has not yet completed execution on that queue

Valid Usage (Implicit)

- VUID-vkImportFenceFdKHR-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkImportFenceFdKHR-pImportFenceFdInfo-parameter
`pImportFenceFdInfo` **must** be a valid pointer to a valid `VkImportFenceFdInfoKHR` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INVALID_EXTERNAL_HANDLE`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkImportFenceFdInfoKHR` structure is defined as:

```
// Provided by VK_KHR_external_fence_fd
typedef struct VkImportFenceFdInfoKHR {
    VkStructureType      sType;
    const void*          pNext;
    VkFence              fence;
    VkFenceImportFlags    flags;
    VkExternalFenceHandleTypeFlagBits handleType;
    int                  fd;
} VkImportFenceFdInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `fence` is the fence into which the payload will be imported.
- `flags` is a bitmask of `VkFenceImportFlagBits` specifying additional parameters for the fence payload import operation.
- `handleType` is a `VkExternalFenceHandleTypeFlagBits` value specifying the type of `fd`.
- `fd` is the external handle to import.

The handle types supported by `handleType` are:

Table 7. Handle Types Supported by `VkImportFenceFdInfoKHR`

Handle Type	Transference	Permanence Supported
<code>VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_FD_BIT</code>	Reference	Temporary,Permanent
<code>VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT</code>	Copy	Temporary

Valid Usage

- VUID-VkImportFenceFdInfoKHR-handleType-01464

handleType **must** be a value included in the [Handle Types Supported by VkImportFenceFdInfoKHR](#) table

- VUID-VkImportFenceFdInfoKHR-fd-01541
fd **must** obey any requirements listed for **handleType** in [external fence handle types compatibility](#)
- VUID-VkImportFenceFdInfoKHR-handleType-07306
If **handleType** refers to a handle type with copy payload transference semantics, **flags** **must** contain `VK_FENCE_IMPORT_TEMPORARY_BIT`

If **handleType** is `VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT`, the special value `-1` for **fd** is treated like a valid sync file descriptor referring to an object that has already signaled. The import operation will succeed and the **VkFence** will have a temporarily imported payload as if a valid file descriptor had been provided.

NOTE

This special behavior for importing an invalid sync file descriptor allows easier interoperability with other system APIs which use the convention that an invalid sync file descriptor represents work that has already completed and does not need to be waited for. It is consistent with the option for implementations to return a `-1` file descriptor when exporting a `VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT` from a **VkFence** which is signaled.

Valid Usage (Implicit)

- VUID-VkImportFenceFdInfoKHR-sType-sType
sType **must** be `VK_STRUCTURE_TYPE_IMPORT_FENCE_FD_INFO_KHR`
- VUID-VkImportFenceFdInfoKHR-pNext-pNext
pNext **must** be `NULL`
- VUID-VkImportFenceFdInfoKHR-fence-parameter
fence **must** be a valid [VkFence](#) handle
- VUID-VkImportFenceFdInfoKHR-flags-parameter
flags **must** be a valid combination of [VkFenceImportFlagBits](#) values
- VUID-VkImportFenceFdInfoKHR-handleType-parameter
handleType **must** be a valid [VkExternalFenceHandleTypeFlagBits](#) value

Host Synchronization

- Host access to **fence** **must** be externally synchronized

To import a fence payload from a **NvSciSyncFence** handle, call:

```
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
VkResult vkImportFenceSciSyncFenceNV(
    VkDevice device,
    const VkImportFenceSciSyncInfoNV* pImportFenceSciSyncInfo);
```

- **device** is the logical device that created the fence.
- **pImportFenceSciSyncInfo** is a pointer to a [VkImportFenceSciSyncInfoNV](#) structure containing parameters of the import operation

Importing a fence payload from **NvSciSyncFence** does not transfer ownership of the handle to the Vulkan implementation. Vulkan will make a copy of **NvSciSyncFence** when importing it. The application **must** release ownership using the NvSciSync API when the handle is no longer needed.

Valid Usage

- VUID-vkImportFenceSciSyncFenceNV-sciSyncImport-05140
[VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncImport](#) and
[VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncFence](#), or
[VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncImport](#) and
[VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncFence](#) **must** be enabled
- VUID-vkImportFenceSciSyncFenceNV-fence-05141
fence **must** not be associated with any queue command that has not yet completed execution on that queue
- VUID-vkImportFenceSciSyncFenceNV-pImportFenceSciSyncInfo-05142
pImportFenceSciSyncInfo->handleType **must** be
[VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV](#)

Valid Usage (Implicit)

- VUID-vkImportFenceSciSyncFenceNV-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkImportFenceSciSyncFenceNV-pImportFenceSciSyncInfo-parameter
pImportFenceSciSyncInfo **must** be a valid pointer to a valid [VkImportFenceSciSyncInfoNV](#) structure

Return Codes

Success

- [VK_SUCCESS](#)

Failure

- [VK_ERROR_INVALID_EXTERNAL_HANDLE](#)
- [VK_ERROR_NOT_PERMITTED](#)

- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To import a fence payload from a `NvSciSyncObj` handle, call:

```
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
VkResult vkImportFenceSciSyncObjNV(
    VkDevice device,
    const VkImportFenceSciSyncInfoNV* pImportFenceSciSyncInfo);
```

- `device` is the logical device that created the fence.
- `pImportFenceSciSyncInfo` is a pointer to a `VkImportFenceSciSyncInfoNV` structure containing parameters of the import operation

Importing a fence payload from a `NvSciSyncObj` does not transfer ownership of the handle to the Vulkan implementation. Vulkan will make a new reference to the `NvSciSyncObj` object when importing it. The application **must** release ownership using the `NvSciSync` API when the handle is no longer needed.

The application **must** not import the same `NvSciSyncObj` with signaler access permissions into multiple instances of `VkFence`, and **must** not import into the same instance from which it was exported.

Valid Usage

- VUID-vkImportFenceSciSyncObjNV-sciSyncImport-05143
`VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncImport` and
`VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncFence`, or
`VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncImport` and
`VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncFence` **must** be enabled
- VUID-vkImportFenceSciSyncObjNV-fence-05144
`fence` **must** not be associated with any queue command that has not yet completed execution on that queue
- VUID-vkImportFenceSciSyncObjNV-pImportFenceSciSyncInfo-05145
`pImportFenceSciSyncInfo->handleType` **must** be
`VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV`

Valid Usage (Implicit)

- VUID-vkImportFenceSciSyncObjNV-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkImportFenceSciSyncObjNV-pImportFenceSciSyncInfo-parameter
`pImportFenceSciSyncInfo` **must** be a valid pointer to a valid `VkImportFenceSciSyncInfoNV` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INVALID_EXTERNAL_HANDLE`
- `VK_ERROR_NOT_PERMITTED`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkImportFenceSciSyncInfoNV` structure is defined as:

```
// Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
typedef struct VkImportFenceSciSyncInfoNV {
    VkStructureType          sType;
    const void*              pNext;
    VkFence                  fence;
    VkExternalFenceHandleTypeFlagBits handleType;
    void*                    handle;
} VkImportFenceSciSyncInfoNV;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `fence` is the fence into which the state will be imported.
- `handleType` specifies the type of `handle`.
- `handle` is the external handle to import.

The handle types supported by `handleType` are:

Table 8. Handle Types Supported by `VkImportFenceSciSyncInfoNV`

Handle Type	Transference	Permanence Supported
<code>VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV</code>	Reference	Permanent
<code>VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV</code>	Copy	Temporary

Valid Usage (Implicit)

- `VUID-VkImportFenceSciSyncInfoNV-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_FENCE_SCI_SYNC_INFO_NV`

- VUID-VkImportFenceSciSyncInfoNV-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkImportFenceSciSyncInfoNV-fence-parameter
`fence` **must** be a valid `VkFence` handle
- VUID-VkImportFenceSciSyncInfoNV-handleType-parameter
`handleType` **must** be a valid `VkExternalFenceHandleTypeFlagBits` value
- VUID-VkImportFenceSciSyncInfoNV-handle-parameter
`handle` **must** be a pointer value

Host Synchronization

- Host access to `fence` **must** be externally synchronized

Bits which **can** be set in

- `VkImportFenceFdInfoKHR::flags`

specifying additional parameters of a fence import operation are:

```
// Provided by VK_VERSION_1_1
typedef enum VkFenceImportFlagBits {
    VK_FENCE_IMPORT_TEMPORARY_BIT = 0x00000001,
} VkFenceImportFlagBits;
```

- `VK_FENCE_IMPORT_TEMPORARY_BIT` specifies that the fence payload will be imported only temporarily, as described in [Importing Fence Payloads](#), regardless of the permanence of `handleType`.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkFenceImportFlags;
```

`VkFenceImportFlags` is a bitmask type for setting a mask of zero or more `VkFenceImportFlagBits`.

7.4. Semaphores

Semaphores are a synchronization primitive that **can** be used to insert a dependency between queue operations or between a queue operation and the host. [Binary semaphores](#) have two states - signaled and unsignaled. [Timeline semaphores](#) have a strictly increasing 64-bit unsigned integer payload and are signaled with respect to a particular reference value. A semaphore **can** be signaled after execution of a queue operation is completed, and a queue operation **can** wait for a semaphore to become signaled before it begins execution. A timeline semaphore **can** additionally be signaled from the host with the `vkSignalSemaphore` command and waited on from the host with the `vkWaitSemaphores` command.

The internal data of a semaphore **may** include a reference to any resources and pending work associated with signal or unsignal operations performed on that semaphore object, collectively referred to as the semaphore's *payload*. Mechanisms to import and export that internal data to and from semaphores are provided [below](#). These mechanisms indirectly enable applications to share semaphore state between two or more semaphores and other synchronization primitives across process and API boundaries.

Semaphores are represented by `VkSemaphore` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSemaphore)
```

To create a semaphore, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateSemaphore(
    VkDevice                                device,
    const VkSemaphoreCreateInfo*            pCreateInfo,
    const VkAllocationCallbacks*           pAllocator,
    VkSemaphore*                            pSemaphore);
```

- `device` is the logical device that creates the semaphore.
- `pCreateInfo` is a pointer to a [VkSemaphoreCreateInfo](#) structure containing information about how the semaphore is to be created.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pSemaphore` is a pointer to a handle in which the resulting semaphore object is returned.

If [VkSemaphoreSciSyncCreateInfoNV](#) is included in the `pNext` chain of [VkSemaphoreCreateInfo](#), [VkSemaphoreSciSyncCreateInfoNV::semaphorePool](#) **may** be accessed any time the resulting semaphore is accessed.

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is `VK_TRUE`, `vkCreateSemaphore` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateSemaphore-device-05068

The number of semaphores currently allocated from `device` plus 1 **must** be less than or equal to the total number of semaphores requested via [VkDeviceObjectReservationCreateInfo::semaphoreRequestCount](#) specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateSemaphore-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkCreateSemaphore-pCreateInfo-parameter
`pCreateInfo` **must** be a valid pointer to a valid `VkSemaphoreCreateInfo` structure
- VUID-vkCreateSemaphore-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateSemaphore-pSemaphore-parameter
`pSemaphore` **must** be a valid pointer to a `VkSemaphore` handle
- VUID-vkCreateSemaphore-device-queuecount
The device **must** have been created with at least `1` queue

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkSemaphoreCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSemaphoreCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkSemaphoreCreateFlags flags;
} VkSemaphoreCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.

Valid Usage

- VUID-VkSemaphoreCreateInfo-pNext-05118
If the `pNext` chain includes `VkExportSemaphoreSciSyncInfoNV`, it **must** also include

`VkSemaphoreTypeCreateInfo` with a `VkSemaphoreTypeCreateInfo::semaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`

- VUID-VkSemaphoreCreateInfo-pNext-05146

If the `pNext` chain includes `VkSemaphoreSciSyncCreateInfoNV`, it **must** also include `VkSemaphoreTypeCreateInfo` with a `VkSemaphoreTypeCreateInfo::semaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`

Valid Usage (Implicit)

- VUID-VkSemaphoreCreateInfo-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO`

- VUID-VkSemaphoreCreateInfo-pNext-pNext

Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkExportSemaphoreCreateInfo`, `VkExportSemaphoreSciSyncInfoNV`, `VkSemaphoreSciSyncCreateInfoNV`, or `VkSemaphoreTypeCreateInfo`

- VUID-VkSemaphoreCreateInfo-sType-unique

The `sType` value of each structure in the `pNext` chain **must** be unique

- VUID-VkSemaphoreCreateInfo-flags-zero bitmask

`flags` **must** be 0

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkSemaphoreCreateFlags;
```

`VkSemaphoreCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

The `VkSemaphoreTypeCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkSemaphoreTypeCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkSemaphoreType    semaphoreType;
    uint64_t           initialValue;
} VkSemaphoreTypeCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `semaphoreType` is a `VkSemaphoreType` value specifying the type of the semaphore.
- `initialValue` is the initial payload value if `semaphoreType` is `VK_SEMAPHORE_TYPE_TIMELINE`.

To create a semaphore of a specific type, add a `VkSemaphoreTypeCreateInfo` structure to the

`VkSemaphoreCreateInfo::pNext` chain.

If no `VkSemaphoreTypeCreateInfo` structure is included in the `pNext` chain of `VkSemaphoreCreateInfo`, then the created semaphore will have a default `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_BINARY`.

If `VkSemaphoreSciSyncCreateInfoNV` structure is included in the `pNext` chain of `VkSemaphoreTypeCreateInfo`, `initialValue` is ignored.

Valid Usage

- VUID-VkSemaphoreTypeCreateInfo-timelineSemaphore-03252
If the `timelineSemaphore` feature is not enabled, `semaphoreType` **must** not equal `VK_SEMAPHORE_TYPE_TIMELINE`
- VUID-VkSemaphoreTypeCreateInfo-semaphoreType-03279
If `semaphoreType` is `VK_SEMAPHORE_TYPE_BINARY`, `initialValue` **must** be zero
- VUID-VkSemaphoreTypeCreateInfo-pNext-05119
If the `pNext` chain includes `VkExportSemaphoreSciSyncInfoNV`, `initialValue` **must** be zero

Valid Usage (Implicit)

- VUID-VkSemaphoreTypeCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO`
- VUID-VkSemaphoreTypeCreateInfo-semaphoreType-parameter
`semaphoreType` **must** be a valid `VkSemaphoreType` value

Possible values of `VkSemaphoreTypeCreateInfo::semaphoreType`, specifying the type of a semaphore, are:

```
// Provided by VK_VERSION_1_2
typedef enum VkSemaphoreType {
    VK_SEMAPHORE_TYPE_BINARY = 0,
    VK_SEMAPHORE_TYPE_TIMELINE = 1,
} VkSemaphoreType;
```

- `VK_SEMAPHORE_TYPE_BINARY` specifies a *binary semaphore* type that has a boolean payload indicating whether the semaphore is currently signaled or unsignaled. When created, the semaphore is in the unsignaled state.
- `VK_SEMAPHORE_TYPE_TIMELINE` specifies a *timeline semaphore* type that has a strictly increasing 64-bit unsigned integer payload indicating whether the semaphore is signaled with respect to a particular reference value. When created, the semaphore payload has the value given by the `initialValue` field of `VkSemaphoreTypeCreateInfo`.

To create a semaphore whose payload **can** be exported to external handles, add a `VkExportSemaphoreCreateInfo` structure to the `pNext` chain of the `VkSemaphoreCreateInfo`

structure. The `VkExportSemaphoreCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExportSemaphoreCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkExternalSemaphoreHandleTypeFlags handleTypes;
} VkExportSemaphoreCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleTypes` is a bitmask of `VkExternalSemaphoreHandleTypeFlagBits` specifying one or more semaphore handle types the application **can** export from the resulting semaphore. The application **can** request multiple handle types for the same semaphore.

Valid Usage

- VUID-VkExportSemaphoreCreateInfo-handleTypes-01124
The bits in `handleTypes` **must** be supported and compatible, as reported by `VkExternalSemaphoreProperties`
- VUID-VkExportSemaphoreCreateInfo-pNext-05120
If the `pNext` chain includes a `VkExportSemaphoreSciSyncInfoNV` structure, the `VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncSemaphore` and `VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncExport` features **must** be enabled

Valid Usage (Implicit)

- VUID-VkExportSemaphoreCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_CREATE_INFO`
- VUID-VkExportSemaphoreCreateInfo-handleTypes-parameter
`handleTypes` **must** be a valid combination of `VkExternalSemaphoreHandleTypeFlagBits` values

To export a POSIX file descriptor representing the payload of a semaphore, call:

```
// Provided by VK_KHR_external_semaphore_fd
VkResult vkGetSemaphoreFdKHR(
    VkDevice          device,
    const VkSemaphoreGetFdInfoKHR* pGetFdInfo,
    int*              pFd);
```

- `device` is the logical device that created the semaphore being exported.
- `pGetFdInfo` is a pointer to a `VkSemaphoreGetFdInfoKHR` structure containing parameters of the

export operation.

- `pFd` will return the file descriptor representing the semaphore payload.

Each call to `vkGetSemaphoreFdKHR` **must** create a new file descriptor and transfer ownership of it to the application. To avoid leaking resources, the application **must** release ownership of the file descriptor when it is no longer needed.

NOTE

Ownership can be released in many ways. For example, the application can call `close()` on the file descriptor, or transfer ownership back to Vulkan by using the file descriptor to import a semaphore payload.

Where supported by the operating system, the implementation **must** set the file descriptor to be closed automatically when an `execve` system call is made.

Exporting a file descriptor from a semaphore **may** have side effects depending on the transference of the specified handle type, as described in [Importing Semaphore State](#).

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetSemaphoreFdKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetSemaphoreFdKHR-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetSemaphoreFdKHR-pGetFdInfo-parameter
`pGetFdInfo` **must** be a valid pointer to a valid `VkSemaphoreGetFdInfoKHR` structure
- VUID-vkGetSemaphoreFdKHR-pFd-parameter
`pFd` **must** be a valid pointer to an `int` value

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkSemaphoreGetFdInfoKHR` structure is defined as:

```
// Provided by VK_KHR_external_semaphore_fd
typedef struct VkSemaphoreGetFdInfoKHR {
    VkStructureType           sType;
    const void*               pNext;
    VkSemaphore               semaphore;
    VkExternalSemaphoreHandleTypeFlagBits handleType;
} VkSemaphoreGetFdInfoKHR;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **semaphore** is the semaphore from which state will be exported.
- **handleType** is a [VkExternalSemaphoreHandleTypeFlagBits](#) value specifying the type of handle requested.

The properties of the file descriptor returned depend on the value of **handleType**. See [VkExternalSemaphoreHandleTypeFlagBits](#) for a description of the properties of the defined external semaphore handle types.

Valid Usage

- VUID-VkSemaphoreGetFdInfoKHR-handleType-01132
handleType **must** have been included in [VkExportSemaphoreCreateInfo::handleTypes](#) when **semaphore**'s current payload was created
- VUID-VkSemaphoreGetFdInfoKHR-semaphore-01133
semaphore **must** not currently have its payload replaced by an imported payload as described below in [Importing Semaphore Payloads](#) unless that imported payload's handle type was included in [VkExternalSemaphoreProperties::exportFromImportedHandleTypes](#) for **handleType**
- VUID-VkSemaphoreGetFdInfoKHR-handleType-01134
If **handleType** refers to a handle type with copy payload transference semantics, as defined below in [Importing Semaphore Payloads](#), there **must** be no queue waiting on **semaphore**
- VUID-VkSemaphoreGetFdInfoKHR-handleType-01135
If **handleType** refers to a handle type with copy payload transference semantics, **semaphore** **must** be signaled, or have an associated [semaphore signal operation](#) pending execution
- VUID-VkSemaphoreGetFdInfoKHR-handleType-01136
handleType **must** be defined as a POSIX file descriptor handle
- VUID-VkSemaphoreGetFdInfoKHR-handleType-03253
If **handleType** refers to a handle type with copy payload transference semantics, **semaphore** **must** have been created with a [VkSemaphoreType](#) of **VK_SEMAPHORE_TYPE_BINARY**
- VUID-VkSemaphoreGetFdInfoKHR-handleType-03254
If **handleType** refers to a handle type with copy payload transference semantics, **semaphore** **must** have an associated semaphore signal operation that has been submitted for execution and any semaphore signal operations on which it depends **must** have also been

Valid Usage (Implicit)

- VUID-VkSemaphoreGetFdInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SEMAPHORE_GET_FD_INFO_KHR`
- VUID-VkSemaphoreGetFdInfoKHR-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkSemaphoreGetFdInfoKHR-semaphore-parameter
`semaphore` **must** be a valid `VkSemaphore` handle
- VUID-VkSemaphoreGetFdInfoKHR-handleType-parameter
`handleType` **must** be a valid `VkExternalSemaphoreHandleTypeFlagBits` value

To specify additional attributes of `NvSciSync` handles exported from a semaphore, add a `VkExportSemaphoreSciSyncInfoNV` structure to the `pNext` chain of the `VkSemaphoreCreateInfo` structure. The `VkExportSemaphoreSciSyncInfoNV` structure is defined as:

```
// Provided by VK_NV_external_sci_sync
typedef struct VkExportSemaphoreSciSyncInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    NvSciSyncAttrList  pAttributes;
} VkExportSemaphoreSciSyncInfoNV;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pAttributes` is an opaque `NvSciSyncAttrList` describing the attributes of the `NvSciSync` object that will be exported.

If `VkExportSemaphoreCreateInfo` is not present in the same `pNext` chain, this structure is ignored. If the `pNext` chain of `VkSemaphoreCreateInfo` includes a `VkExportSemaphoreCreateInfo` structure with a `NvSciSync` `handleType`, but either `VkExportSemaphoreSciSyncInfoNV` is not included in the `pNext` chain, or it is included but `pAttributes` is `NULL`, `vkCreateSemaphore` will return `VK_ERROR_INITIALIZATION_FAILED`.

The `pAttributes` **must** be a reconciled `NvSciSyncAttrList`. Before exporting a `NvSciSync` handle, the application **must** use the `vkGetPhysicalDeviceSciSyncAttributesNV` command to obtain the unreconciled `NvSciSyncAttrList` and then use the `NvSciSync` API to reconcile it.

Valid Usage

- VUID-VkExportSemaphoreSciSyncInfoNV-pAttributes-05121
`pAttributes` **must** be a reconciled `NvSciSyncAttrList`

Valid Usage (Implicit)

- VUID-VkExportSemaphoreSciSyncInfoNV-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_SCI_SYNC_INFO_NV`

To export a `NvSciSyncObj` handle representing the payload of a semaphore, call:

```
// Provided by VK_NV_external_sci_sync
VkResult vkGetSemaphoreSciSyncObjNV(
    VkDevice device,
    const VkSemaphoreGetSciSyncInfoNV* pGetSciSyncInfo,
    void* pHandle);
```

- `device` is the logical device that created the semaphore being exported.
- `pGetSciSyncInfo` is a pointer to a `VkSemaphoreGetSciSyncInfoNV` structure containing parameters of the export operation.
- `pHandle` will return the `NvSciSyncObj` representing the semaphore payload.

Each call to `vkGetSemaphoreSciSyncObjNV` will duplicate the underlying `NvSciSyncObj` and transfer the ownership of the `NvSciSyncObj` handle to the application. To avoid leaking resources, the application **must** release ownership of the `NvSciSyncObj` when it is no longer needed.

Valid Usage

- VUID-vkGetSemaphoreSciSyncObjNV-sciSyncSemaphore-05147
`VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncSemaphore` **must** be enabled

Valid Usage (Implicit)

- VUID-vkGetSemaphoreSciSyncObjNV-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetSemaphoreSciSyncObjNV-pGetSciSyncInfo-parameter
`pGetSciSyncInfo` **must** be a valid pointer to a valid `VkSemaphoreGetSciSyncInfoNV` structure
- VUID-vkGetSemaphoreSciSyncObjNV-pHandle-parameter
`pHandle` **must** be a pointer value

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INVALID_EXTERNAL_HANDLE`
- `VK_ERROR_NOT_PERMITTED`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkSemaphoreGetSciSyncInfoNV` structure is defined as:

```
// Provided by VK_NV_external_sci_sync
typedef struct VkSemaphoreGetSciSyncInfoNV {
    VkStructureType          sType;
    const void*              pNext;
    VkSemaphore              semaphore;
    VkExternalSemaphoreHandleTypeFlagBits handleType;
} VkSemaphoreGetSciSyncInfoNV;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `semaphore` is the semaphore from which state will be exported.
- `handleType` is the type of `NvSciSync` handle (`NvSciSyncObj`) representing the semaphore that will be exported.

Valid Usage

- VUID-VkSemaphoreGetSciSyncInfoNV-handleType-05122
`handleType` **must** be `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV`
- VUID-VkSemaphoreGetSciSyncInfoNV-semaphore-05123
`semaphore` **must** have been created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`
- VUID-VkSemaphoreGetSciSyncInfoNV-semaphore-05129
`semaphore` **must** have been created with `VkExportSemaphoreSciSyncInfoNV` included `pNext` chain of `VkSemaphoreCreateInfo`, or previously imported by `vkImportSemaphoreSciSyncObjNV`

Valid Usage (Implicit)

- VUID-VkSemaphoreGetSciSyncInfoNV-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SEMAPHORE_GET_SCI_SYNC_INFO_NV`
- VUID-VkSemaphoreGetSciSyncInfoNV-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkSemaphoreGetSciSyncInfoNV-semaphore-parameter

semaphore must be a valid [VkSemaphore](#) handle

- VUID-VkSemaphoreGetSciSyncInfoNV-handleType-parameter **handleType** must be a valid [VkExternalSemaphoreHandleTypeFlagBits](#) value

The [VkSemaphoreSciSyncCreateInfoNV](#) structure is defined as:

```
// Provided by VK_NV_external_sci_sync2
typedef struct VkSemaphoreSciSyncCreateInfoNV {
    VkStructureType      sType;
    const void*          pNext;
    VkSemaphoreSciSyncPoolNV semaphorePool;
    const NvSciSyncFence* pFence;
} VkSemaphoreSciSyncCreateInfoNV;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is `NULL` or a pointer to a structure extending this structure.
- **semaphorePool** is a [VkSemaphoreSciSyncPoolNV](#) handle.
- **pFence** is a pointer to a [NvSciSyncFence](#).

When [VkSemaphoreSciSyncCreateInfoNV](#) is included in [VkSemaphoreCreateInfo::pNext](#) chain, the semaphore is created from the [VkSemaphoreSciSyncPoolNV](#) handle that represents a [NvSciSyncObj](#) with one or more primitives. The [VkSemaphoreSciSyncCreateInfoNV::pFence](#) parameter provides the information to select the corresponding primitive represented by this semaphore. When a [NvSciSyncObj](#) with signaler permissions is imported to [VkSemaphoreSciSyncPoolNV](#), it only supports one primitive and [VkSemaphoreSciSyncCreateInfoNV::pFence](#) must be in the cleared state.

Valid Usage

- VUID-VkSemaphoreSciSyncCreateInfoNV-sciSyncSemaphore2-05148
The [VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncSemaphore2](#) feature must be enabled

Valid Usage (Implicit)

- VUID-VkSemaphoreSciSyncCreateInfoNV-sType-sType
sType must be `VK_STRUCTURE_TYPE_SEMAPHORE_SCI_SYNC_CREATE_INFO_NV`
- VUID-VkSemaphoreSciSyncCreateInfoNV-semaphorePool-parameter
semaphorePool must be a valid [VkSemaphoreSciSyncPoolNV](#) handle
- VUID-VkSemaphoreSciSyncCreateInfoNV-pFence-parameter
pFence must be a valid pointer to a valid [NvSciSyncFence](#) value

To destroy a semaphore, call:

```
// Provided by VK_VERSION_1_0
void vkDestroySemaphore(
    VkDevice          device,
    VkSemaphore        semaphore,
    const VkAllocationCallbacks* pAllocator);
```

- **device** is the logical device that destroys the semaphore.
- **semaphore** is the handle of the semaphore to destroy.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.

If **semaphore** was created with [VkSemaphoreSciSyncCreateInfoNV](#) present in the [VkSemaphoreCreateInfo::pNext](#) chain, **semaphore** **can** be destroyed immediately after all batches that refer to it are submitted. Otherwise, all submitted batches that refer to **semaphore** **must** have completed execution before it can be destroyed.

Valid Usage

- VUID-vkDestroySemaphore-semaphore-05149
If **semaphore** was not created with [VkSemaphoreSciSyncCreateInfoNV](#) present in the [VkSemaphoreCreateInfo::pNext](#) chain when it was created, all submitted batches that refer to **semaphore** **must** have completed execution

Valid Usage (Implicit)

- VUID-vkDestroySemaphore-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkDestroySemaphore-semaphore-parameter
If **semaphore** is not [VK_NULL_HANDLE](#), **semaphore** **must** be a valid [VkSemaphore](#) handle
- VUID-vkDestroySemaphore-pAllocator-null
pAllocator **must** be [NULL](#)
- VUID-vkDestroySemaphore-semaphore-parent
If **semaphore** is a valid handle, it **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to **semaphore** **must** be externally synchronized

7.4.1. Semaphore SciSync Pools

A semaphore SciSync pool is used to represent a [NvSciSyncObj](#) with one or more primitives.

Semaphore SciSync pools are represented by `VkSemaphoreSciSyncPoolNV` handles:

```
// Provided by VK_NV_external_sci_sync2
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSemaphoreSciSyncPoolNV)
```

Semaphore SciSync pools **cannot** be freed [SCID-4].

If `VkPhysicalDeviceVulkanSC10Properties::deviceDestroyFreesMemory` is `VK_TRUE`, the memory is returned to the system and the reference to the `NvSciSyncObj` that was imported is released when the device is destroyed. Otherwise, it **may** not be returned to the system until the process is terminated.

To import a `NvSciSyncObj` with multiple primitives, use `vkCreateSemaphoreSciSyncPoolNV` to reserve a semaphore pool to map the multiple semaphores allocated by `NvSciSyncObj`. Then create a `VkSemaphore` from the semaphore pool using the index provided by the `NvSciSyncFence` when chaining the `VkSemaphoreSciSyncCreateInfoNV` structure to `VkSemaphoreCreateInfo`.

To create a `VkSemaphoreSciSyncPoolNV`, call:

```
// Provided by VK_NV_external_sci_sync2
VkResult vkCreateSemaphoreSciSyncPoolNV(
    VkDevice device,
    const VkSemaphoreSciSyncPoolCreateInfoNV* pCreateInfo,
    const VkAllocationCallbacks* pAllocator,
    VkSemaphoreSciSyncPoolNV* pSemaphorePool);
```

- `device` is the logical device that creates the semaphore pool.
- `pCreateInfo` is a pointer to a `VkSemaphoreSciSyncPoolCreateInfoNV` structure containing information about the semaphore SciSync pool being created.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pSemaphorePool` is a pointer to a handle in which the resulting semaphore pool object is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateSemaphoreSciSyncPoolNV` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateSemaphoreSciSyncPoolNV-sciSyncSemaphore2-05151
The `VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncSemaphore2` feature **must** be enabled
- VUID-vkCreateSemaphoreSciSyncPoolNV-device-05068
The number of semaphore pools currently allocated from `device` plus 1 **must** be less than or equal to the total number of semaphore pools requested via `VkDeviceSemaphoreSciSyncPoolReservationCreateInfoNV::semaphoreSciSyncPoolRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateSemaphoreSciSyncPoolNV-device-parameter **device** **must** be a valid [VkDevice](#) handle
- VUID-vkCreateSemaphoreSciSyncPoolNV-pCreateInfo-parameter **pCreateInfo** **must** be a valid pointer to a valid [VkSemaphoreSciSyncPoolCreateInfoNV](#) structure
- VUID-vkCreateSemaphoreSciSyncPoolNV-pAllocator-null **pAllocator** **must** be `NULL`
- VUID-vkCreateSemaphoreSciSyncPoolNV-pSemaphorePool-parameter **pSemaphorePool** **must** be a valid pointer to a [VkSemaphoreSciSyncPoolNV](#) handle
- VUID-vkCreateSemaphoreSciSyncPoolNV-device-queuecount
The device **must** have been created with at least `1` queue

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkSemaphoreSciSyncPoolCreateInfoNV` structure is defined as:

```
// Provided by VK_NV_external_sci_sync2
typedef struct VkSemaphoreSciSyncPoolCreateInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    NvSciSyncObj       handle;
} VkSemaphoreSciSyncPoolCreateInfoNV;
```

- `sType` is a [VkStructureType](#) value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handle` is an external [NvSciSyncObj](#) to import.

During [vkCreateSemaphoreSciSyncPoolNV](#), the external `NvSciSyncObj` is imported to `VkSemaphoreSciSyncPoolNV`. The import does not transfer the ownership of the `NvSciSyncObj` to the implementation, but will increment the reference count of that object. The application **must** delete other references of the original `NvSciSyncObj` using [NvSciSync APIs](#) when it is no longer needed.

Applications **must** not import the same `NvSciSyncObj` with signaler access permissions to multiple instances of `VkSemaphoreSciSyncPoolNV`.

Valid Usage

- VUID-VkSemaphoreSciSyncPoolCreateInfoNV-handle-05152
`handle` **must** be a valid `NvSciSyncObj`

Valid Usage (Implicit)

- VUID-VkSemaphoreSciSyncPoolCreateInfoNV-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SEMAPHORE_SCI_SYNC_POOL_CREATE_INFO_NV`
- VUID-VkSemaphoreSciSyncPoolCreateInfoNV-pNext-pNext
`pNext` **must** be `NULL`

7.4.2. Semaphore Signaling

When a batch is submitted to a queue via a [queue submission](#), and it includes semaphores to be signaled, it defines a memory dependency on the batch, and defines *semaphore signal operations* which set the semaphores to the signaled state.

In case of semaphores created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE` the semaphore is considered signaled with respect to the counter value set to be signaled as specified in [VkTimelineSemaphoreSubmitInfo](#) or [VkSemaphoreSignalInfo](#).

The first [synchronization scope](#) includes every command submitted in the same batch. In the case of [vkQueueSubmit2KHR](#), the first synchronization scope is limited to the pipeline stage specified by [VkSemaphoreSubmitInfo::stageMask](#). Semaphore signal operations that are defined by [vkQueueSubmit](#) or [vkQueueSubmit2KHR](#) additionally include all commands that occur earlier in [submission order](#). Semaphore signal operations that are defined by [vkQueueSubmit](#) or [vkQueueSubmit2KHR](#) additionally include in the first synchronization scope any semaphore and fence signal operations that occur earlier in [signal operation order](#).

The second [synchronization scope](#) includes only the semaphore signal operation.

The first [access scope](#) includes all memory access performed by the device.

The second [access scope](#) is empty.

7.4.3. Semaphore Waiting

When a batch is submitted to a queue via a [queue submission](#), and it includes semaphores to be waited on, it defines a memory dependency between prior semaphore signal operations and the batch, and defines *semaphore wait operations*.

Such semaphore wait operations set the semaphores created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_BINARY` to the unsignaled state. In case of semaphores created with a

`VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE` a prior semaphore signal operation defines a memory dependency with a semaphore wait operation if the value the semaphore is signaled with is greater than or equal to the value the semaphore is waited with, thus the semaphore will continue to be considered signaled with respect to the counter value waited on as specified in `VkTimelineSemaphoreSubmitInfo`.

The first `synchronization scope` includes one `semaphore signal operation` for each semaphore waited on by this batch. The specific signal operation waited on for each semaphore **must** meet the following criteria:

- for binary semaphores, the signal operation is either earlier in `submission order` on the same queue, or is submitted by a command whose host operation happens-before this batch is submitted on the host
- for binary semaphores, no wait operation exists that happens-after the signal operation and happens-before this wait operation
- the signal operation is not guaranteed to happen-after the semaphore wait operation in this batch
- for timeline semaphores, the signal value is greater than or equal to the wait value

If multiple semaphore signal operations meet these criteria, any of those operations **may** be included in the first `synchronization scope`. When waiting on a binary semaphore, applications **must** ensure that exactly one semaphore signal operation meets these criteria.

The second `synchronization scope` includes every command submitted in the same batch. In the case of `vkQueueSubmit`, the second synchronization scope is limited to operations on the pipeline stages determined by the `destination stage mask` specified by the corresponding element of `pWaitDstStageMask`. In the case of `vkQueueSubmit2KHR`, the second synchronization scope is limited to the pipeline stage specified by `VkSemaphoreSubmitInfo::stageMask`. Also, in the case of either `vkQueueSubmit2KHR` or `vkQueueSubmit`, the second synchronization scope additionally includes all commands that occur later in `submission order`.

The first `access scope` is empty.

The second `access scope` includes all memory access performed by the device.

The semaphore wait operation happens-after the first set of operations in the execution dependency, and happens-before the second set of operations in the execution dependency.

NOTE

Unlike timeline semaphores, fences or events, waiting for a binary semaphore also unsignals that semaphore when the wait completes. Applications **must** ensure that between two such wait operations, the semaphore is signaled again, with execution dependencies used to ensure these occur in order. Binary semaphore waits and signals should thus occur in discrete 1:1 pairs.

NOTE

A common scenario for using `pWaitDstStageMask` with values other than `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT` is when synchronizing a window system presentation operation against subsequent command buffers which render the next frame. In this case, a presentation image **must** not be overwritten until the

presentation operation completes, but other pipeline stages **can** execute without waiting. A mask of `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT` prevents subsequent color attachment writes from executing until the semaphore signals. Some implementations **may** be able to execute transfer operations and/or pre-rasterization work before the semaphore is signaled.

If an image layout transition needs to be performed on a presentable image before it is used in a framebuffer, that **can** be performed as the first operation submitted to the queue after acquiring the image, and **should** not prevent other work from overlapping with the presentation operation. For example, a `VkImageMemoryBarrier` could use:

- `srcStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT`
- `srcAccessMask = 0`
- `dstStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT`
- `dstAccessMask = VK_ACCESS_COLOR_ATTACHMENT_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT.`
- `oldLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR`
- `newLayout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`

Alternatively, `oldLayout` **can** be `VK_IMAGE_LAYOUT_UNDEFINED`, if the image's contents need not be preserved.

This barrier accomplishes a dependency chain between previous presentation operations and subsequent color attachment output operations, with the layout transition performed in between, and does not introduce a dependency between previous work and any [pre-rasterization shader stages](#). More precisely, the semaphore signals after the presentation operation completes, the semaphore wait stalls the `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT` stage, and there is a dependency from that same stage to itself with the layout transition performed in between.

7.4.4. Semaphore State Requirements for Wait Operations

Before waiting on a semaphore, the application **must** ensure the semaphore is in a valid state for a wait operation. Specifically, when a [semaphore wait operation](#) is submitted to a queue:

- A binary semaphore **must** be signaled, or have an associated [semaphore signal operation](#) that is pending execution.
- Any [semaphore signal operations](#) on which the pending binary semaphore signal operation depends **must** also be completed or pending execution.
- There **must** be no other queue waiting on the same binary semaphore when the operation executes.

7.4.5. Host Operations on Semaphores

In addition to [semaphore signal operations](#) and [semaphore wait operations](#) submitted to device queues, timeline semaphores support the following host operations:

- Query the current counter value of the semaphore using the [vkGetSemaphoreCounterValue](#) command.
- Wait for a set of semaphores to reach particular counter values using the [vkWaitSemaphores](#) command.
- Signal the semaphore with a particular counter value from the host using the [vkSignalSemaphore](#) command.

To query the current counter value of a semaphore created with a [VkSemaphoreType](#) of [VK_SEMAPHORE_TYPE_TIMELINE](#) from the host, call:

```
// Provided by VK_VERSION_1_2
VkResult vkGetSemaphoreCounterValue(
    VkDevice          device,
    VkSemaphore       semaphore,
    uint64_t*         pValue);
```

- [device](#) is the logical device that owns the semaphore.
- [semaphore](#) is the handle of the semaphore to query.
- [pValue](#) is a pointer to a 64-bit integer value in which the current counter value of the semaphore is returned.

NOTE

If a [queue submission](#) command is pending execution, then the value returned by this command **may** immediately be out of date.

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is [VK_TRUE](#), [vkGetSemaphoreCounterValue](#) **must** not return [VK_ERROR_OUT_OF_HOST_MEMORY](#).

Valid Usage

- VUID-vkGetSemaphoreCounterValue-semaphore-03255
[semaphore](#) **must** have been created with a [VkSemaphoreType](#) of [VK_SEMAPHORE_TYPE_TIMELINE](#)

Valid Usage (Implicit)

- VUID-vkGetSemaphoreCounterValue-device-parameter
[device](#) **must** be a valid [VkDevice](#) handle
- VUID-vkGetSemaphoreCounterValue-semaphore-parameter
[semaphore](#) **must** be a valid [VkSemaphore](#) handle

- VUID-vkGetSemaphoreCounterValue-pValue-parameter
`pValue` **must** be a valid pointer to a `uint64_t` value
- VUID-vkGetSemaphoreCounterValue-semaphore-parent
`semaphore` **must** have been created, allocated, or retrieved from `device`

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To wait for a set of semaphores created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE` to reach particular counter values on the host, call:

```
// Provided by VK_VERSION_1_2
VkResult vkWaitSemaphores(
    VkDevice          device,
    const VkSemaphoreWaitInfo* pWaitInfo,
    uint64_t          timeout);
```

- `device` is the logical device that owns the semaphores.
- `pWaitInfo` is a pointer to a `VkSemaphoreWaitInfo` structure containing information about the wait condition.
- `timeout` is the timeout period in units of nanoseconds. `timeout` is adjusted to the closest value allowed by the implementation-dependent timeout accuracy, which **may** be substantially longer than one nanosecond, and **may** be longer than the requested period.

If the condition is satisfied when `vkWaitSemaphores` is called, then `vkWaitSemaphores` returns immediately. If the condition is not satisfied at the time `vkWaitSemaphores` is called, then `vkWaitSemaphores` will block and wait until the condition is satisfied or the `timeout` has expired, whichever is sooner.

If `timeout` is zero, then `vkWaitSemaphores` does not wait, but simply returns information about the current state of the semaphores. `VK_TIMEOUT` will be returned in this case if the condition is not satisfied, even though no actual wait was performed.

If the condition is satisfied before the `timeout` has expired, `vkWaitSemaphores` returns `VK_SUCCESS`. Otherwise, `vkWaitSemaphores` returns `VK_TIMEOUT` after the `timeout` has expired.

If device loss occurs (see [Lost Device](#)) before the timeout has expired, `vkWaitSemaphores` **must** return in finite time with either `VK_SUCCESS` or `VK_ERROR_DEVICE_LOST`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkWaitSemaphores` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkWaitSemaphores-device-parameter `device` **must** be a valid [VkDevice](#) handle
- VUID-vkWaitSemaphores-pWaitInfo-parameter `pWaitInfo` **must** be a valid pointer to a valid [VkSemaphoreWaitInfo](#) structure

Return Codes

Success

- `VK_SUCCESS`
- `VK_TIMEOUT`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkSemaphoreWaitInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkSemaphoreWaitInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkSemaphoreWaitFlags flags;
    uint32_t           semaphoreCount;
    const VkSemaphore* pSemaphores;
    const uint64_t*    pValues;
} VkSemaphoreWaitInfo;
```

- `sType` is a [VkStructureType](#) value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of [VkSemaphoreWaitFlagBits](#) specifying additional parameters for the semaphore wait operation.

- `semaphoreCount` is the number of semaphores to wait on.
- `pSemaphores` is a pointer to an array of `semaphoreCount` semaphore handles to wait on.
- `pValues` is a pointer to an array of `semaphoreCount` timeline semaphore values.

Valid Usage

- VUID-VkSemaphoreWaitInfo-pSemaphores-03256
All of the elements of `pSemaphores` **must** reference a semaphore that was created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`
- VUID-VkSemaphoreWaitInfo-pSemaphores-05124
If any of the semaphores in `pSemaphores` have `NvSciSyncObj` as payload, application **must** calculate the corresponding timeline semaphore values in `pValues` by calling `NvSciSync APIs`

Valid Usage (Implicit)

- VUID-VkSemaphoreWaitInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SEMAPHORE_WAIT_INFO`
- VUID-VkSemaphoreWaitInfo-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkSemaphoreWaitInfo-flags-parameter
`flags` **must** be a valid combination of `VkSemaphoreWaitFlagBits` values
- VUID-VkSemaphoreWaitInfo-pSemaphores-parameter
`pSemaphores` **must** be a valid pointer to an array of `semaphoreCount` valid `VkSemaphore` handles
- VUID-VkSemaphoreWaitInfo-pValues-parameter
`pValues` **must** be a valid pointer to an array of `semaphoreCount uint64_t` values
- VUID-VkSemaphoreWaitInfo-semaphoreCount-arraylength
`semaphoreCount` **must** be greater than `0`

Bits which **can** be set in `VkSemaphoreWaitInfo::flags`, specifying additional parameters of a semaphore wait operation, are:

```
// Provided by VK_VERSION_1_2
typedef enum VkSemaphoreWaitFlagBits {
    VK_SEMAPHORE_WAIT_ANY_BIT = 0x00000001,
} VkSemaphoreWaitFlagBits;
```

- `VK_SEMAPHORE_WAIT_ANY_BIT` specifies that the semaphore wait condition is that at least one of the semaphores in `VkSemaphoreWaitInfo::pSemaphores` has reached the value specified by the corresponding element of `VkSemaphoreWaitInfo::pValues`. If `VK_SEMAPHORE_WAIT_ANY_BIT` is not set, the semaphore wait condition is that all of the semaphores in `VkSemaphoreWaitInfo::pSemaphores`

have reached the value specified by the corresponding element of `VkSemaphoreWaitInfo::pValues`.

```
// Provided by VK_VERSION_1_2
typedef VkFlags VkSemaphoreWaitFlags;
```

`VkSemaphoreWaitFlags` is a bitmask type for setting a mask of zero or more `VkSemaphoreWaitFlagBits`.

To signal a semaphore created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE` with a particular counter value, on the host, call:

```
// Provided by VK_VERSION_1_2
VkResult vkSignalSemaphore(
    VkDevice device,
    const VkSemaphoreSignalInfo* pSignalInfo);
```

- `device` is the logical device that owns the semaphore.
- `pSignalInfo` is a pointer to a `VkSemaphoreSignalInfo` structure containing information about the signal operation.

When `vkSignalSemaphore` is executed on the host, it defines and immediately executes a *semaphore signal operation* which sets the timeline semaphore to the given value.

The first synchronization scope is defined by the host execution model, but includes execution of `vkSignalSemaphore` on the host and anything that happened-before it.

The second synchronization scope is empty.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkSignalSemaphore` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkSignalSemaphore-device-parameter `device` **must** be a valid `VkDevice` handle
- VUID-vkSignalSemaphore-pSignalInfo-parameter `pSignalInfo` **must** be a valid pointer to a valid `VkSemaphoreSignalInfo` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkSemaphoreSignalInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkSemaphoreSignalInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkSemaphore        semaphore;
    uint64_t           value;
} VkSemaphoreSignalInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `semaphore` is the handle of the semaphore to signal.
- `value` is the value to signal.

Valid Usage

- VUID-VkSemaphoreSignalInfo-semaphore-03257
`semaphore` **must** have been created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`
- VUID-VkSemaphoreSignalInfo-value-03258
`value` **must** have a value greater than the current value of the semaphore
- VUID-VkSemaphoreSignalInfo-value-03259
`value` **must** be less than the value of any pending semaphore signal operations
- VUID-VkSemaphoreSignalInfo-value-03260
`value` **must** have a value which does not differ from the current value of the semaphore or the value of any outstanding semaphore wait or signal operation on `semaphore` by more than `maxTimelineSemaphoreValueDifference`
- VUID-VkSemaphoreSignalInfo-semaphores-05125
If `semaphores` has `NvSciSyncObj` as payload, application **must** calculate the corresponding timeline semaphore value in `value` by calling `NvSciSync` APIs

Valid Usage (Implicit)

- VUID-VkSemaphoreSignalInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SEMAPHORE_SIGNAL_INFO`
- VUID-VkSemaphoreSignalInfo-pNext-pNext

pNext **must** be NULL

- VUID-VkSemaphoreSignalInfo-semaphore-parameter semaphore **must** be a valid [VkSemaphore](#) handle

7.4.6. Importing Semaphore Payloads

Applications **can** import a semaphore payload into an existing semaphore using an external semaphore handle. The effects of the import operation will be either temporary or permanent, as specified by the application. If the import is temporary, the implementation **must** restore the semaphore to its prior permanent state after submitting the next semaphore wait operation. Performing a subsequent temporary import on a semaphore before performing a semaphore wait has no effect on this requirement; the next wait submitted on the semaphore **must** still restore its last permanent state. A permanent payload import behaves as if the target semaphore was destroyed, and a new semaphore was created with the same handle but the imported payload. Because importing a semaphore payload temporarily or permanently detaches the existing payload from a semaphore, similar usage restrictions to those applied to [vkDestroySemaphore](#) are applied to any command that imports a semaphore payload. Which of these import types is used is referred to as the import operation's *permanence*. Each handle type supports either one or both types of permanence.

The implementation **must** perform the import operation by either referencing or copying the payload referred to by the specified external semaphore handle, depending on the handle's type. The import method used is referred to as the handle type's *transference*. When using handle types with reference transference, importing a payload to a semaphore adds the semaphore to the set of all semaphores sharing that payload. This set includes the semaphore from which the payload was exported. Semaphore signaling and waiting operations performed on any semaphore in the set **must** behave as if the set were a single semaphore. Importing a payload using handle types with copy transference creates a duplicate copy of the payload at the time of import, but makes no further reference to it. Semaphore signaling and waiting operations performed on the target of copy imports **must** not affect any other semaphore or payload.

Export operations have the same transference as the specified handle type's import operations. Additionally, exporting a semaphore payload to a handle with copy transference has the same side effects on the source semaphore's payload as executing a semaphore wait operation. If the semaphore was using a temporarily imported payload, the semaphore's prior permanent payload will be restored.

NOTE

The permanence and transference of handle types can be found in:

- [Handle Types Supported by VkImportSemaphoreFdInfoKHR](#)

[External synchronization](#) allows implementations to modify an object's internal state, i.e. payload, without internal synchronization. However, for semaphores sharing a payload across processes, satisfying the external synchronization requirements of [VkSemaphore](#) parameters as if all semaphores in the set were the same object is sometimes infeasible. Satisfying the [wait operation state requirements](#) would similarly require impractical coordination or levels of trust between processes. Therefore, these constraints only apply to a specific semaphore handle, not to its

payload. For distinct semaphore objects which share a payload, if the semaphores are passed to separate queue submission commands concurrently, behavior will be as if the commands were called in an arbitrary sequential order. If the [wait operation state requirements](#) are violated for the shared payload by a queue submission command, or if a signal operation is queued for a shared payload that is already signaled or has a pending signal operation, effects **must** be limited to one or more of the following:

- Returning `VK_ERROR_INITIALIZATION_FAILED` from the command which resulted in the violation.
- Losing the logical device on which the violation occurred immediately or at a future time, resulting in a `VK_ERROR_DEVICE_LOST` error from subsequent commands, including the one causing the violation.
- Continuing execution of the violating command or operation as if the semaphore wait completed successfully after an implementation-dependent timeout. In this case, the state of the payload becomes undefined, and future operations on semaphores sharing the payload will be subject to these same rules. The semaphore **must** be destroyed or have its payload replaced by an import operation to again have a well-defined state.

NOTE

These rules allow processes to synchronize access to shared memory without trusting each other. However, such processes must still be cautious not to use the shared semaphore for more than synchronizing access to the shared memory. For example, a process should not use a shared semaphore as part of an execution dependency chain that, when complete, leads to objects being destroyed, if it does not trust other processes sharing the semaphore payload.

When a semaphore is using an imported payload, its `VkExportSemaphoreCreateInfo::handleTypes` value is specified when creating the semaphore from which the payload was exported, rather than specified when creating the semaphore. Additionally, `VkExternalSemaphoreProperties::exportFromImportedHandleTypes` restricts which handle types **can** be exported from such a semaphore based on the specific handle type used to import the current payload. Passing a semaphore to `vkAcquireNextImageKHR` is equivalent to temporarily importing a semaphore payload to that semaphore.

NOTE

Because the exportable handle types of an imported semaphore correspond to its current imported payload, and `vkAcquireNextImageKHR` behaves the same as a temporary import operation for which the source semaphore is opaque to the application, applications have no way of determining whether any external handle types **can** be exported from a semaphore in this state. Therefore, applications **must** not attempt to export external handles from semaphores using a temporarily imported payload from `vkAcquireNextImageKHR`.

When importing a semaphore payload, it is the responsibility of the application to ensure the external handles meet all valid usage requirements. However, implementations **must** perform sufficient validation of external handles to ensure that the operation results in a valid semaphore which will not cause program termination, device loss, queue stalls, or corruption of other resources when used as allowed according to its import parameters, and excepting those side effects allowed for violations of the [valid semaphore state for wait operations](#) rules. If the external handle provided does not meet these requirements, the implementation **must** fail the semaphore

payload import operation with the error code `VK_ERROR_INVALID_EXTERNAL_HANDLE`.

In addition, when importing a semaphore payload that is not compatible with the payload type corresponding to the `VkSemaphoreType` the semaphore was created with, the implementation **may** fail the semaphore payload import operation with the error code `VK_ERROR_INVALID_EXTERNAL_HANDLE`.

NOTE

As the introduction of the external semaphore handle type `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT` predates that of timeline semaphores, support for importing semaphore payloads from external handles of that type into semaphores created (implicitly or explicitly) with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_BINARY` is preserved for backwards compatibility. However, applications **should** prefer importing such handle types into semaphores created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`.

To import a semaphore payload from a POSIX file descriptor, call:

```
// Provided by VK_KHR_external_semaphore_fd
VkResult vkImportSemaphoreFdKHR(
    VkDevice device,
    const VkImportSemaphoreFdInfoKHR* pImportSemaphoreFdInfo);
```

- `device` is the logical device that created the semaphore.
- `pImportSemaphoreFdInfo` is a pointer to a `VkImportSemaphoreFdInfoKHR` structure specifying the semaphore and import parameters.

Importing a semaphore payload from a file descriptor transfers ownership of the file descriptor from the application to the Vulkan implementation. The application **must** not perform any operations on the file descriptor after a successful import.

Applications **can** import the same semaphore payload into multiple instances of Vulkan, into the same instance from which it was exported, and multiple times into a given Vulkan instance.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkImportSemaphoreFdKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkImportSemaphoreFdKHR-semaphore-01142
`semaphore` **must** not be associated with any queue command that has not yet completed execution on that queue

Valid Usage (Implicit)

- VUID-vkImportSemaphoreFdKHR-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkImportSemaphoreFdKHR-pImportSemaphoreFdInfo-parameter

`pImportSemaphoreFdInfo` **must** be a valid pointer to a valid `VkImportSemaphoreFdInfoKHR` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INVALID_EXTERNAL_HANDLE`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkImportSemaphoreFdInfoKHR` structure is defined as:

```
// Provided by VK_KHR_external_semaphore_fd
typedef struct VkImportSemaphoreFdInfoKHR {
    VkStructureType          sType;
    const void*              pNext;
    VkSemaphore              semaphore;
    VkSemaphoreImportFlags    flags;
    VkExternalSemaphoreHandleTypeFlagBits handleType;
    int                      fd;
} VkImportSemaphoreFdInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `semaphore` is the semaphore into which the payload will be imported.
- `flags` is a bitmask of `VkSemaphoreImportFlagBits` specifying additional parameters for the semaphore payload import operation.
- `handleType` is a `VkExternalSemaphoreHandleTypeFlagBits` value specifying the type of `fd`.
- `fd` is the external handle to import.

The handle types supported by `handleType` are:

Table 9. Handle Types Supported by `VkImportSemaphoreFdInfoKHR`

Handle Type	Transference	Permanence Supported
<code>VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT</code>	Reference	Temporary,Permanent
<code>VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT</code>	Copy	Temporary

Valid Usage

- VUID-VkImportSemaphoreFdInfoKHR-handleType-01143
`handleType` **must** be a value included in the [Handle Types Supported by VkImportSemaphoreFdInfoKHR](#) table
- VUID-VkImportSemaphoreFdInfoKHR-fd-01544
`fd` **must** obey any requirements listed for `handleType` in [external semaphore handle types compatibility](#)
- VUID-VkImportSemaphoreFdInfoKHR-handleType-03263
If `handleType` is `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT`, the `VkSemaphoreCreateInfo::flags` field **must** match that of the semaphore from which `fd` was exported
- VUID-VkImportSemaphoreFdInfoKHR-handleType-07307
If `handleType` refers to a handle type with copy payload transference semantics, `flags` **must** contain `VK_SEMAPHORE_IMPORT_TEMPORARY_BIT`
- VUID-VkImportSemaphoreFdInfoKHR-handleType-03264
If `handleType` is `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT`, the `VkSemaphoreTypeCreateInfo::semaphoreType` field **must** match that of the semaphore from which `fd` was exported
- VUID-VkImportSemaphoreFdInfoKHR-flags-03323
If `flags` contains `VK_SEMAPHORE_IMPORT_TEMPORARY_BIT`, the `VkSemaphoreTypeCreateInfo::semaphoreType` field of the semaphore from which `fd` was exported **must** not be `VK_SEMAPHORE_TYPE_TIMELINE`

If `handleType` is `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT`, the special value `-1` for `fd` is treated like a valid sync file descriptor referring to an object that has already signaled. The import operation will succeed and the `VkSemaphore` will have a temporarily imported payload as if a valid file descriptor had been provided.

NOTE

This special behavior for importing an invalid sync file descriptor allows easier interoperability with other system APIs which use the convention that an invalid sync file descriptor represents work that has already completed and does not need to be waited for. It is consistent with the option for implementations to return a `-1` file descriptor when exporting a `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT` from a `VkSemaphore` which is signaled.

Valid Usage (Implicit)

- VUID-VkImportSemaphoreFdInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_FD_INFO_KHR`
- VUID-VkImportSemaphoreFdInfoKHR-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkImportSemaphoreFdInfoKHR-semaphore-parameter

semaphore **must** be a valid [VkSemaphore](#) handle

- VUID-VkImportSemaphoreFdInfoKHR-flags-parameter
flags **must** be a valid combination of [VkSemaphoreImportFlagBits](#) values
- VUID-VkImportSemaphoreFdInfoKHR-handleType-parameter
handleType **must** be a valid [VkExternalSemaphoreHandleTypeFlagBits](#) value

Host Synchronization

- Host access to **semaphore** **must** be externally synchronized

To import a semaphore payload from a [NvSciSyncObj](#), call:

```
// Provided by VK_NV_external_sci_sync
VkResult vkImportSemaphoreSciSyncObjNV(
    VkDevice device,
    const VkImportSemaphoreSciSyncInfoNV* pImportSemaphoreSciSyncInfo);
```

- **device** is the logical device that created the semaphore.
- **pImportSemaphoreSciSyncInfo** is a pointer to a [VkImportSemaphoreSciSyncInfoNV](#) structure containing parameters of the import operation

Importing a semaphore payload from [NvSciSyncObj](#) does not transfer ownership of the handle to the Vulkan implementation. When importing [NvSciSyncObj](#), Vulkan will make a new reference to that object, the application **must** release its ownership using [NvSciSync APIs](#) when that ownership is no longer needed.

Application **must** not import the same [NvSciSyncObj](#) with signaler access permissions into multiple instances of [VkSemaphore](#), and **must** not import into the same instance from which it was exported.

Valid Usage

- VUID-vkImportSemaphoreSciSyncObjNV-sciSyncImport-05155
[VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncImport](#) and
[VkPhysicalDeviceExternalSciSyncFeaturesNV::sciSyncSemaphore](#) **must** be enabled

Valid Usage (Implicit)

- VUID-vkImportSemaphoreSciSyncObjNV-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkImportSemaphoreSciSyncObjNV-pImportSemaphoreSciSyncInfo-parameter
pImportSemaphoreSciSyncInfo **must** be a valid pointer to a valid [VkImportSemaphoreSciSyncInfoNV](#) structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INVALID_EXTERNAL_HANDLE`
- `VK_ERROR_NOT_PERMITTED`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkImportSemaphoreSciSyncInfoNV` structure is defined as:

```
// Provided by VK_NV_external_sci_sync
typedef struct VkImportSemaphoreSciSyncInfoNV {
    VkStructureType          sType;
    const void*              pNext;
    VkSemaphore              semaphore;
    VkExternalSemaphoreHandleTypeFlagBits handleType;
    void*                    handle;
} VkImportSemaphoreSciSyncInfoNV;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `semaphore` is the semaphore into which the payload will be imported.
- `handleType` specifies the type of `handle`.
- `handle` is the external handle to import.

The handle types supported by `handleType` are:

Table 10. Handle Types Supported by `VkImportSemaphoreSciSyncInfoNV`

Handle Type	Transference	Permanence Supported
<code>VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SCI_SYNC_OBJECT_NV</code>	Reference	Permanent

Valid Usage

- VUID-VkImportSemaphoreSciSyncInfoNV-handleType-05126
`handleType` **must** be a value included in the [Handle Types Supported by `VkImportSemaphoreSciSyncInfoNV`](#) table
- VUID-VkImportSemaphoreSciSyncInfoNV-semaphore-05127

`semaphore` **must** have been created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`

- VUID-VkImportSemaphoreSciSyncInfoNV-semaphore-05128
`semaphore` **must** not be associated with any queue command that has not yet completed execution on that queue

Valid Usage (Implicit)

- VUID-VkImportSemaphoreSciSyncInfoNV-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_SCI_SYNC_INFO_NV`
- VUID-VkImportSemaphoreSciSyncInfoNV-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkImportSemaphoreSciSyncInfoNV-semaphore-parameter
`semaphore` **must** be a valid `VkSemaphore` handle
- VUID-VkImportSemaphoreSciSyncInfoNV-handleType-parameter
`handleType` **must** be a valid `VkExternalSemaphoreHandleTypeFlagBits` value
- VUID-VkImportSemaphoreSciSyncInfoNV-handle-parameter
`handle` **must** be a pointer value

Host Synchronization

- Host access to `semaphore` **must** be externally synchronized

Bits which **can** be set in

- `VkImportSemaphoreFdInfoKHR::flags`

specifying additional parameters of a semaphore import operation are:

```
// Provided by VK_VERSION_1_1
typedef enum VkSemaphoreImportFlagBits {
    VK_SEMAPHORE_IMPORT_TEMPORARY_BIT = 0x00000001,
} VkSemaphoreImportFlagBits;
```

These bits have the following meanings:

- `VK_SEMAPHORE_IMPORT_TEMPORARY_BIT` specifies that the semaphore payload will be imported only temporarily, as described in [Importing Semaphore Payloads](#), regardless of the permanence of `handleType`.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkSemaphoreImportFlags;
```

`VkSemaphoreImportFlags` is a bitmask type for setting a mask of zero or more `VkSemaphoreImportFlagBits`.

7.5. Events

Events are a synchronization primitive that **can** be used to insert a fine-grained dependency between commands submitted to the same queue, or between the host and a queue. Events **must** not be used to insert a dependency between commands submitted to different queues. Events have two states - signaled and unsignaled. An application **can** signal or unsignal an event either on the host or on the device. A device **can** be made to wait for an event to become signaled before executing further operations. No command exists to wait for an event to become signaled on the host, but the current state of an event **can** be queried.

Events are represented by `VkEvent` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkEvent)
```

To create an event, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateEvent(
    VkDevice                                device,
    const VkEventCreateInfo*                pCreateInfo,
    const VkAllocationCallbacks*            pAllocator,
    VkEvent*                                 pEvent);
```

- `device` is the logical device that creates the event.
- `pCreateInfo` is a pointer to a `VkEventCreateInfo` structure containing information about how the event is to be created.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pEvent` is a pointer to a handle in which the resulting event object is returned.

When created, the event object is in the unsignaled state.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateEvent` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateEvent-device-09672
`device` **must** support at least one queue family with one of the `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` capabilities
- VUID-vkCreateEvent-device-05068

The number of events currently allocated from `device` plus 1 **must** be less than or equal to the total number of events requested via `VkDeviceObjectReservationCreateInfo::eventRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateEvent-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkCreateEvent-pCreateInfo-parameter
`pCreateInfo` **must** be a valid pointer to a valid `VkEventCreateInfo` structure
- VUID-vkCreateEvent-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateEvent-pEvent-parameter
`pEvent` **must** be a valid pointer to a `VkEvent` handle
- VUID-vkCreateEvent-device-queuecount
The device **must** have been created with at least 1 queue

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkEventCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkEventCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkEventCreateFlags flags;
} VkEventCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkEventCreateFlagBits` defining additional creation parameters.

Valid Usage (Implicit)

- VUID-VkEventCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_EVENT_CREATE_INFO`
- VUID-VkEventCreateInfo-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkEventCreateInfo-flags-parameter
`flags` **must** be a valid combination of `VkEventCreateFlagBits` values

```
// Provided by VK_VERSION_1_0
typedef enum VkEventCreateFlagBits {
    VK_EVENT_CREATE_DEVICE_ONLY_BIT = 0x00000001,
    // Provided by VK_KHR_synchronization2
    VK_EVENT_CREATE_DEVICE_ONLY_BIT_KHR = VK_EVENT_CREATE_DEVICE_ONLY_BIT,
} VkEventCreateFlagBits;
```

- `VK_EVENT_CREATE_DEVICE_ONLY_BIT` specifies that host event commands will not be used with this event.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkEventCreateFlags;
```

`VkEventCreateFlags` is a bitmask type for setting a mask of `VkEventCreateFlagBits`.

To destroy an event, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyEvent(
    VkDevice          device,
    VkEvent           event,
    const VkAllocationCallbacks* pAllocator);
```

- `device` is the logical device that destroys the event.
- `event` is the handle of the event to destroy.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage

- VUID-vkDestroyEvent-event-01145
All submitted commands that refer to `event` **must** have completed execution

Valid Usage (Implicit)

- VUID-vkDestroyEvent-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkDestroyEvent-event-parameter
If **event** is not [VK_NULL_HANDLE](#), **event** **must** be a valid [VkEvent](#) handle
- VUID-vkDestroyEvent-pAllocator-null
pAllocator **must** be [NULL](#)
- VUID-vkDestroyEvent-event-parent
If **event** is a valid handle, it **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to **event** **must** be externally synchronized

To query the state of an event from the host, call:

```
// Provided by VK_VERSION_1_0
VkResult vkGetEventStatus(
    VkDevice          device,
    VkEvent           event);
```

- **device** is the logical device that owns the event.
- **event** is the handle of the event to query.

Upon success, **vkGetEventStatus** returns the state of the event object with the following return codes:

Table 11. Event Object Status Codes

Status	Meaning
VK_EVENT_SET	The event specified by event is signaled.
VK_EVENT_RESET	The event specified by event is unsignaled.

If a **vkCmdSetEvent** or **vkCmdResetEvent** command is in a command buffer that is in the [pending state](#), then the value returned by this command **may** immediately be out of date.

The state of an event **can** be updated by the host. The state of the event is immediately changed, and subsequent calls to **vkGetEventStatus** will return the new state. If an event is already in the requested state, then updating it to the same state has no effect.

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is [VK_TRUE](#), **vkGetEventStatus** **must** not return [VK_ERROR_OUT_OF_HOST_MEMORY](#).

Valid Usage

- VUID-vkGetEventStatus-event-03940
event **must** not have been created with `VK_EVENT_CREATE_DEVICE_ONLY_BIT`

Valid Usage (Implicit)

- VUID-vkGetEventStatus-device-parameter
device **must** be a valid `VkDevice` handle
- VUID-vkGetEventStatus-event-parameter
event **must** be a valid `VkEvent` handle
- VUID-vkGetEventStatus-event-parent
event **must** have been created, allocated, or retrieved from **device**

Return Codes

Success

- `VK_EVENT_RESET`
- `VK_EVENT_SET`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To set the state of an event to signaled from the host, call:

```
// Provided by VK_VERSION_1_0
VkResult vkSetEvent(
    VkDevice          device,
    VkEvent           event);
```

- **device** is the logical device that owns the event.
- **event** is the event to set.

When `vkSetEvent` is executed on the host, it defines an *event signal operation* which sets the event to the signaled state.

If **event** is already in the signaled state when `vkSetEvent` is executed, then `vkSetEvent` has no effect,

and no event signal operation occurs.

NOTE

If a command buffer is waiting for an event to be signaled from the host, the application must signal the event before submitting the command buffer, as described in the [queue forward progress](#) section.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkSetEvent` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkSetEvent-event-03941
`event` **must** not have been created with `VK_EVENT_CREATE_DEVICE_ONLY_BIT`
- VUID-vkSetEvent-event-09543
`event` **must** not be waited on by a command buffer in the [pending state](#)

Valid Usage (Implicit)

- VUID-vkSetEvent-device-parameter
`device` **must** be a valid [VkDevice](#) handle
- VUID-vkSetEvent-event-parameter
`event` **must** be a valid [VkEvent](#) handle
- VUID-vkSetEvent-event-parent
`event` **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `event` **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To set the state of an event to unsignaled from the host, call:

```
// Provided by VK_VERSION_1_0
VkResult vkResetEvent(
    VkDevice          device,
    VkEvent            event);
```

- **device** is the logical device that owns the event.
- **event** is the event to reset.

When **vkResetEvent** is executed on the host, it defines an *event unsignal operation* which resets the event to the unsignaled state.

If **event** is already in the unsignaled state when **vkResetEvent** is executed, then **vkResetEvent** has no effect, and no event unsignal operation occurs.

Valid Usage

- VUID-vkResetEvent-event-03821
There **must** be an execution dependency between **vkResetEvent** and the execution of any **vkCmdWaitEvents** that includes **event** in its **pEvents** parameter
- VUID-vkResetEvent-event-03822
There **must** be an execution dependency between **vkResetEvent** and the execution of any **vkCmdWaitEvents2KHR** that includes **event** in its **pEvents** parameter
- VUID-vkResetEvent-event-03823
event **must** not have been created with **VK_EVENT_CREATE_DEVICE_ONLY_BIT**

Valid Usage (Implicit)

- VUID-vkResetEvent-device-parameter
device **must** be a valid **VkDevice** handle
- VUID-vkResetEvent-event-parameter
event **must** be a valid **VkEvent** handle
- VUID-vkResetEvent-event-parent
event **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to **event** **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The state of an event **can** also be updated on the device by commands inserted in command buffers.

To signal an event from a device, call:

```
// Provided by VK_KHR_synchronization2
void vkCmdSetEvent2KHR(
    VkCommandBuffer          commandBuffer,
    VkEvent                  event,
    const VkDependencyInfo*  pDependencyInfo);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `event` is the event that will be signaled.
- `pDependencyInfo` is a pointer to a `VkDependencyInfo` structure defining the first scopes of this operation.

When `vkCmdSetEvent2KHR` is submitted to a queue, it defines the first half of memory dependencies defined by `pDependencyInfo`, as well as an event signal operation which sets the event to the signaled state. A memory dependency is defined between the event signal operation and commands that occur earlier in submission order.

The first `synchronization scope` and `access scope` are defined by the union of all the memory dependencies defined by `pDependencyInfo`, and are applied to all operations that occur earlier in `submission order`. `Queue family ownership transfers` and `image layout transitions` defined by `pDependencyInfo` are also included in the first scopes.

The second `synchronization scope` includes only the event signal operation, and any `queue family ownership transfers` and `image layout transitions` defined by `pDependencyInfo`.

The second `access scope` includes only `queue family ownership transfers` and `image layout transitions`.

Future `vkCmdWaitEvents2KHR` commands rely on all values of each element in `pDependencyInfo` matching exactly with those used to signal the corresponding event. `vkCmdWaitEvents` **must** not be used to wait on the result of a signal operation defined by `vkCmdSetEvent2KHR`.

NOTE

The extra information provided by [vkCmdSetEvent2KHR](#) compared to [vkCmdSetEvent](#) allows implementations to more efficiently schedule the operations required to satisfy the requested dependencies. With [vkCmdSetEvent](#), the full dependency information is not known until [vkCmdWaitEvents](#) is recorded, forcing implementations to insert the required operations at that point and not before.

If [event](#) is already in the signaled state when [vkCmdSetEvent2KHR](#) is executed on the device, then [vkCmdSetEvent2KHR](#) has no effect, no event signal operation occurs, and no dependency is generated.

Valid Usage

- VUID-vkCmdSetEvent2-synchronization2-03824
The [synchronization2](#) feature **must** be enabled
- VUID-vkCmdSetEvent2-dependencyFlags-03825
The [dependencyFlags](#) member of [pDependencyInfo](#) **must** be 0
- VUID-vkCmdSetEvent2-srcStageMask-09391
The [srcStageMask](#) member of any element of the [pMemoryBarriers](#), [pBufferMemoryBarriers](#), or [pImageMemoryBarriers](#) members of [pDependencyInfo](#) **must** not include [VK_PIPELINE_STAGE_2_HOST_BIT](#)
- VUID-vkCmdSetEvent2-dstStageMask-09392
The [dstStageMask](#) member of any element of the [pMemoryBarriers](#), [pBufferMemoryBarriers](#), or [pImageMemoryBarriers](#) members of [pDependencyInfo](#) **must** not include [VK_PIPELINE_STAGE_2_HOST_BIT](#)
- VUID-vkCmdSetEvent2-commandBuffer-03826
The current device mask of [commandBuffer](#) **must** include exactly one physical device
- VUID-vkCmdSetEvent2-srcStageMask-03827
The [srcStageMask](#) member of any element of the [pMemoryBarriers](#), [pBufferMemoryBarriers](#), or [pImageMemoryBarriers](#) members of [pDependencyInfo](#) **must** only include pipeline stages valid for the queue family that was used to create the command pool that [commandBuffer](#) was allocated from
- VUID-vkCmdSetEvent2-dstStageMask-03828
The [dstStageMask](#) member of any element of the [pMemoryBarriers](#), [pBufferMemoryBarriers](#), or [pImageMemoryBarriers](#) members of [pDependencyInfo](#) **must** only include pipeline stages valid for the queue family that was used to create the command pool that [commandBuffer](#) was allocated from

Valid Usage (Implicit)

- VUID-vkCmdSetEvent2-commandBuffer-parameter
[commandBuffer](#) **must** be a valid [VkCommandBuffer](#) handle
- VUID-vkCmdSetEvent2-event-parameter
[event](#) **must** be a valid [VkEvent](#) handle

- VUID-vkCmdSetEvent2-pDependencyInfo-parameter
`pDependencyInfo` **must** be a valid pointer to a valid `VkDependencyInfo` structure
- VUID-vkCmdSetEvent2-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdSetEvent2-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdSetEvent2-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdSetEvent2-commonparent
Both of `commandBuffer`, and `event` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT	Synchronization

The `VkDependencyInfo` structure is defined as:

```
typedef struct VkDependencyInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkDependencyFlags     dependencyFlags;
    uint32_t              memoryBarrierCount;
    const VkMemoryBarrier2* pMemoryBarriers;
    uint32_t              bufferMemoryBarrierCount;
    const VkBufferMemoryBarrier2* pBufferMemoryBarriers;
    uint32_t              imageMemoryBarrierCount;
    const VkImageMemoryBarrier2* pImageMemoryBarriers;
} VkDependencyInfo;
```

```
// Provided by VK_KHR_synchronization2
typedef VkDependencyInfo VkDependencyInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `dependencyFlags` is a bitmask of `VkDependencyFlagBits` specifying how execution and memory dependencies are formed.
- `memoryBarrierCount` is the length of the `pMemoryBarriers` array.
- `pMemoryBarriers` is a pointer to an array of `VkMemoryBarrier2` structures defining memory dependencies between any memory accesses.
- `bufferMemoryBarrierCount` is the length of the `pBufferMemoryBarriers` array.
- `pBufferMemoryBarriers` is a pointer to an array of `VkBufferMemoryBarrier2` structures defining memory dependencies between buffer ranges.
- `imageMemoryBarrierCount` is the length of the `pImageMemoryBarriers` array.
- `pImageMemoryBarriers` is a pointer to an array of `VkImageMemoryBarrier2` structures defining memory dependencies between image subresources.

This structure defines a set of [memory dependencies](#), as well as [queue family ownership transfer operations](#) and [image layout transitions](#).

Each member of `pMemoryBarriers`, `pBufferMemoryBarriers`, and `pImageMemoryBarriers` defines a separate [memory dependency](#).

Valid Usage

- VUID-VkDependencyInfo-pMemoryBarriers-10605
For each element of `pMemoryBarriers`, the `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkDependencyInfo-pMemoryBarriers-10606
For each element of `pMemoryBarriers`, `pNext` **must** be `NULL`

Valid Usage (Implicit)

- VUID-VkDependencyInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEPENDENCY_INFO`
- VUID-VkDependencyInfo-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkDependencyInfo-dependencyFlags-parameter
`dependencyFlags` **must** be a valid combination of `VkDependencyFlagBits` values
- VUID-VkDependencyInfo-pMemoryBarriers-parameter
If `memoryBarrierCount` is not 0, `pMemoryBarriers` **must** be a valid pointer to an array of

`memoryBarrierCount` valid `VkMemoryBarrier2` structures

- VUID-VkDependencyInfo-pBufferMemoryBarriers-parameter
If `bufferMemoryBarrierCount` is not 0, `pBufferMemoryBarriers` **must** be a valid pointer to an array of `bufferMemoryBarrierCount` valid `VkBufferMemoryBarrier2` structures
- VUID-VkDependencyInfo-pImageMemoryBarriers-parameter
If `imageMemoryBarrierCount` is not 0, `pImageMemoryBarriers` **must** be a valid pointer to an array of `imageMemoryBarrierCount` valid `VkImageMemoryBarrier2` structures

To set the state of an event to signaled from a device, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetEvent(
    VkCommandBuffer          commandBuffer,
    VkEvent                  event,
    VkPipelineStageFlags     stageMask);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `event` is the event that will be signaled.
- `stageMask` specifies the `source stage mask` used to determine the first `synchronization scope`.

`vkCmdSetEvent` behaves identically to `vkCmdSetEvent2KHR`, except that it does not define an access scope, and **must** only be used with `vkCmdWaitEvents`, not `vkCmdWaitEvents2KHR`.

Valid Usage

- VUID-vkCmdSetEvent-stageMask-04090
If the `geometryShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-vkCmdSetEvent-stageMask-04091
If the `tessellationShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdSetEvent-stageMask-07319
If the `attachmentFragmentShadingRate` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdSetEvent-stageMask-03937
If the `synchronization2` feature is not enabled, `stageMask` **must** not be 0
- VUID-vkCmdSetEvent-stageMask-06457
Any pipeline stage included in `stageMask` **must** be supported by the capabilities of the queue family specified by the `queueFamilyIndex` member of the `VkCommandPoolCreateInfo` structure that was used to create the `VkCommandPool` that `commandBuffer` was allocated from, as specified in the [table of supported pipeline stages](#)
- VUID-vkCmdSetEvent-stageMask-01149

`stageMask` **must** not include `VK_PIPELINE_STAGE_HOST_BIT`

- VUID-vkCmdSetEvent-commandBuffer-01152

The current device mask of `commandBuffer` **must** include exactly one physical device

Valid Usage (Implicit)

- VUID-vkCmdSetEvent-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetEvent-event-parameter
`event` **must** be a valid `VkEvent` handle
- VUID-vkCmdSetEvent-stageMask-parameter
`stageMask` **must** be a valid combination of `VkPipelineStageFlagBits` values
- VUID-vkCmdSetEvent-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetEvent-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdSetEvent-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdSetEvent-commonparent
Both of `commandBuffer`, and `event` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT	Synchronization

To unsignal the event from a device, call:

```
// Provided by VK_KHR_synchronization2
void vkCmdResetEvent2KHR(
    VkCommandBuffer          commandBuffer,
    VkEvent                  event,
    VkPipelineStageFlags2    stageMask);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `event` is the event that will be unsignaled.
- `stageMask` is a `VkPipelineStageFlags2` mask of pipeline stages used to determine the first `synchronization scope`.

When `vkCmdResetEvent2KHR` is submitted to a queue, it defines an execution dependency on commands that were submitted before it, and defines an event unsignal operation which resets the event to the unsignaled state.

The first `synchronization scope` includes all commands that occur earlier in `submission order`. The synchronization scope is limited to operations by `stageMask` or stages that are `logically earlier` than `stageMask`.

The second `synchronization scope` includes only the event unsignal operation.

If `event` is already in the unsignaled state when `vkCmdResetEvent2KHR` is executed on the device, then this command has no effect, no event unsignal operation occurs, and no execution dependency is generated.

Valid Usage

- VUID-vkCmdResetEvent2-stageMask-03929
If the `geometryShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`
- VUID-vkCmdResetEvent2-stageMask-03930
If the `tessellationShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdResetEvent2-stageMask-07317
If the `attachmentFragmentShadingRate` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdResetEvent2-synchronization2-03829
The `synchronization2` feature **must** be enabled
- VUID-vkCmdResetEvent2-stageMask-03830
`stageMask` **must** not include `VK_PIPELINE_STAGE_2_HOST_BIT`
- VUID-vkCmdResetEvent2-event-03831
There **must** be an execution dependency between `vkCmdResetEvent2KHR` and the execution of any `vkCmdWaitEvents` that includes `event` in its `pEvents` parameter

- VUID-vkCmdResetEvent2-event-03832
There **must** be an execution dependency between `vkCmdResetEvent2KHR` and the execution of any `vkCmdWaitEvents2KHR` that includes `event` in its `pEvents` parameter
- VUID-vkCmdResetEvent2-commandBuffer-03833
`commandBuffer`'s current device mask **must** include exactly one physical device

Valid Usage (Implicit)

- VUID-vkCmdResetEvent2-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdResetEvent2-event-parameter
`event` **must** be a valid `VkEvent` handle
- VUID-vkCmdResetEvent2-stageMask-parameter
`stageMask` **must** be a valid combination of `VkPipelineStageFlagBits2` values
- VUID-vkCmdResetEvent2-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdResetEvent2-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdResetEvent2-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdResetEvent2-commonparent
Both of `commandBuffer`, and `event` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT	Synchronization

To set the state of an event to unsignaled from a device, call:

```
// Provided by VK_VERSION_1_0
void vkCmdResetEvent(
    VkCommandBuffer          commandBuffer,
    VkEvent                  event,
    VkPipelineStageFlags     stageMask);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `event` is the event that will be unsignaled.
- `stageMask` is a bitmask of `VkPipelineStageFlagBits` specifying the `source stage mask` used to determine when the `event` is unsignaled.

`vkCmdResetEvent` behaves identically to `vkCmdResetEvent2KHR`.

Valid Usage

- VUID-vkCmdResetEvent-stageMask-04090
If the `geometryShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-vkCmdResetEvent-stageMask-04091
If the `tessellationShader` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdResetEvent-stageMask-07319
If the `attachmentFragmentShadingRate` feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdResetEvent-stageMask-03937
If the `synchronization2` feature is not enabled, `stageMask` **must** not be 0
- VUID-vkCmdResetEvent-stageMask-06458
Any pipeline stage included in `stageMask` **must** be supported by the capabilities of the queue family specified by the `queueFamilyIndex` member of the `VkCommandPoolCreateInfo` structure that was used to create the `VkCommandPool` that `commandBuffer` was allocated from, as specified in the [table of supported pipeline stages](#)
- VUID-vkCmdResetEvent-stageMask-01153
`stageMask` **must** not include `VK_PIPELINE_STAGE_HOST_BIT`
- VUID-vkCmdResetEvent-event-03834
There **must** be an execution dependency between `vkCmdResetEvent` and the execution of any `vkCmdWaitEvents` that includes `event` in its `pEvents` parameter
- VUID-vkCmdResetEvent-event-03835
There **must** be an execution dependency between `vkCmdResetEvent` and the execution of any `vkCmdWaitEvents2KHR` that includes `event` in its `pEvents` parameter
- VUID-vkCmdResetEvent-commandBuffer-01157

`commandBuffer`'s current device mask **must** include exactly one physical device

Valid Usage (Implicit)

- VUID-vkCmdResetEvent-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdResetEvent-event-parameter
`event` **must** be a valid `VkEvent` handle
- VUID-vkCmdResetEvent-stageMask-parameter
`stageMask` **must** be a valid combination of `VkPipelineStageFlagBits` values
- VUID-vkCmdResetEvent-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdResetEvent-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdResetEvent-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdResetEvent-commonparent
Both of `commandBuffer`, and `event` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT	Synchronization

To wait for one or more events to enter the signaled state on a device, call:

```
// Provided by VK_KHR_synchronization2
void vkCmdWaitEvents2KHR(
    VkCommandBuffer                commandBuffer,
    uint32_t                       eventCount,
    const VkEvent*                 pEvents,
    const VkDependencyInfo*        pDependencyInfos);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `eventCount` is the length of the `pEvents` array.
- `pEvents` is a pointer to an array of `eventCount` events to wait on.
- `pDependencyInfos` is a pointer to an array of `eventCount` `VkDependencyInfo` structures, defining the second `synchronization scope`.

When `vkCmdWaitEvents2KHR` is submitted to a queue, it inserts memory dependencies according to the elements of `pDependencyInfos` and each corresponding element of `pEvents`. `vkCmdWaitEvents2KHR` **must** not be used to wait on event signal operations occurring on other queues, or signal operations executed by `vkCmdSetEvent`.

The first `synchronization scope` and `access scope` of each memory dependency defined by any element `i` of `pDependencyInfos` are applied to operations that occurred earlier in `submission order` than the last event signal operation on element `i` of `pEvents`.

Signal operations for an event at index `i` are only included if:

- The event was signaled by a `vkCmdSetEvent2KHR` command that occurred earlier in `submission order` with a `dependencyInfo` parameter exactly equal to the element of `pDependencyInfos` at index `i`; or
- The event was created without `VK_EVENT_CREATE_DEVICE_ONLY_BIT`, and the first `synchronization scope` defined by the element of `pDependencyInfos` at index `i` only includes host operations (`VK_PIPELINE_STAGE_2_HOST_BIT`).

The second `synchronization scope` and `access scope` of each memory dependency defined by any element `i` of `pDependencyInfos` are applied to operations that occurred later in `submission order` than `vkCmdWaitEvents2KHR`.

NOTE

`vkCmdWaitEvents2KHR` is used with `vkCmdSetEvent2KHR` to define a memory dependency between two sets of action commands, roughly in the same way as pipeline barriers, but split into two commands such that work between the two **may** execute unhindered.

NOTE

Applications should be careful to avoid race conditions when using events. There is no direct ordering guarantee between `vkCmdSetEvent2KHR` and `vkCmdResetEvent2KHR`, `vkCmdResetEvent`, or `vkCmdSetEvent`. Another execution dependency (e.g. a pipeline barrier or semaphore with `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`) is needed to prevent such a race condition.

Valid Usage

- VUID-vkCmdWaitEvents2-synchronization2-03836
The **synchronization2** feature **must** be enabled
- VUID-vkCmdWaitEvents2-pEvents-03837
Members of **pEvents** **must** not have been signaled by **vkCmdSetEvent**
- VUID-vkCmdWaitEvents2-pEvents-10788
For any element *i* of **pEvents**, if that event is signaled by **vkCmdSetEvent2KHR**, that command's **dependencyInfo** parameter **must** be exactly equal to the *i*th element of **pDependencyInfos**
- VUID-vkCmdWaitEvents2-pEvents-03839
For any element *i* of **pEvents**, if that event is signaled by **vkSetEvent**, barriers in the *i*th element of **pDependencyInfos** **must** include only host operations in their first **synchronization scope**
- VUID-vkCmdWaitEvents2-pEvents-03840
For any element *i* of **pEvents**, if barriers in the *i*th element of **pDependencyInfos** include only host operations, the *i*th element of **pEvents** **must** be signaled before **vkCmdWaitEvents2KHR** is executed
- VUID-vkCmdWaitEvents2-pEvents-03841
For any element *i* of **pEvents**, if barriers in the *i*th element of **pDependencyInfos** do not include host operations, the *i*th element of **pEvents** **must** be signaled by a corresponding **vkCmdSetEvent2KHR** that occurred earlier in **submission order**
- VUID-vkCmdWaitEvents2-srcStageMask-03842
The **srcStageMask** member of any element of the **pMemoryBarriers**, **pBufferMemoryBarriers**, or **pImageMemoryBarriers** members of **pDependencyInfos** **must** only include pipeline stages valid for the queue family that was used to create the command pool that **commandBuffer** was allocated from
- VUID-vkCmdWaitEvents2-dstStageMask-03843
The **dstStageMask** member of any element of the **pMemoryBarriers**, **pBufferMemoryBarriers**, or **pImageMemoryBarriers** members of **pDependencyInfos** **must** only include pipeline stages valid for the queue family that was used to create the command pool that **commandBuffer** was allocated from
- VUID-vkCmdWaitEvents2-dependencyFlags-10394
The **dependencyFlags** member of any element of **pDependencyInfo** **must** not include any of the following bits:
 - **VK_DEPENDENCY_BY_REGION_BIT**
 - **VK_DEPENDENCY_DEVICE_GROUP_BIT**
 - **VK_DEPENDENCY_VIEW_LOCAL_BIT**
- VUID-vkCmdWaitEvents2-dependencyFlags-03844
If this command is called inside a render pass instance, the **srcStageMask** member of any element of the **pMemoryBarriers**, **pBufferMemoryBarriers**, or **pImageMemoryBarriers** members of **pDependencyInfos** **must** not include **VK_PIPELINE_STAGE_2_HOST_BIT**

- VUID-vkCmdWaitEvents2-commandBuffer-03846
`commandBuffer`'s current device mask **must** include exactly one physical device

Valid Usage (Implicit)

- VUID-vkCmdWaitEvents2-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdWaitEvents2-pEvents-parameter
`pEvents` **must** be a valid pointer to an array of `eventCount` valid `VkEvent` handles
- VUID-vkCmdWaitEvents2-pDependencyInfos-parameter
`pDependencyInfos` **must** be a valid pointer to an array of `eventCount` valid `VkDependencyInfo` structures
- VUID-vkCmdWaitEvents2-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdWaitEvents2-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdWaitEvents2-eventCount-arraylength
`eventCount` **must** be greater than 0
- VUID-vkCmdWaitEvents2-commonparent
Both of `commandBuffer`, and the elements of `pEvents` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT	Synchronization

To wait for one or more events to enter the signaled state on a device, call:

```
// Provided by VK_VERSION_1_0
void vkCmdWaitEvents(
    VkCommandBuffer                commandBuffer,
    uint32_t                       eventCount,
    const VkEvent*                 pEvents,
    VkPipelineStageFlags           srcStageMask,
    VkPipelineStageFlags           dstStageMask,
    uint32_t                       memoryBarrierCount,
    const VkMemoryBarrier*         pMemoryBarriers,
    uint32_t                       bufferMemoryBarrierCount,
    const VkBufferMemoryBarrier*   pBufferMemoryBarriers,
    uint32_t                       imageMemoryBarrierCount,
    const VkImageMemoryBarrier*    pImageMemoryBarriers);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `eventCount` is the length of the `pEvents` array.
- `pEvents` is a pointer to an array of event object handles to wait on.
- `srcStageMask` is a bitmask of `VkPipelineStageFlagBits` specifying the [source stage mask](#).
- `dstStageMask` is a bitmask of `VkPipelineStageFlagBits` specifying the [destination stage mask](#).
- `memoryBarrierCount` is the length of the `pMemoryBarriers` array.
- `pMemoryBarriers` is a pointer to an array of `VkMemoryBarrier` structures.
- `bufferMemoryBarrierCount` is the length of the `pBufferMemoryBarriers` array.
- `pBufferMemoryBarriers` is a pointer to an array of `VkBufferMemoryBarrier` structures.
- `imageMemoryBarrierCount` is the length of the `pImageMemoryBarriers` array.
- `pImageMemoryBarriers` is a pointer to an array of `VkImageMemoryBarrier` structures.

`vkCmdWaitEvents` is largely similar to `vkCmdWaitEvents2KHR`, but **can** only wait on signal operations defined by `vkCmdSetEvent`. As `vkCmdSetEvent` does not define any access scopes, `vkCmdWaitEvents` defines the first access scope for each event signal operation in addition to its own access scopes.

NOTE	Since <code>vkCmdSetEvent</code> does not have any dependency information beyond a stage mask, implementations do not have the same opportunity to perform availability and visibility operations or image layout transitions in advance as they do with <code>vkCmdSetEvent2KHR</code> and <code>vkCmdWaitEvents2KHR</code> .

When `vkCmdWaitEvents` is submitted to a queue, it defines a memory dependency between prior event signal operations on the same queue or the host, and subsequent commands. `vkCmdWaitEvents` **must** not be used to wait on event signal operations occurring on other queues.

The first synchronization scope only includes event signal operations that operate on members of `pEvents`, and the operations that happened-before the event signal operations. Event signal operations performed by `vkCmdSetEvent` that occur earlier in [submission order](#) are included in the first synchronization scope, if the [logically latest](#) pipeline stage in their `stageMask` parameter is [logically earlier](#) than or equal to the [logically latest](#) pipeline stage in `srcStageMask`. Event signal

operations performed by `vkSetEvent` are only included in the first synchronization scope if `VK_PIPELINE_STAGE_HOST_BIT` is included in `srcStageMask`.

The second `synchronization scope` includes all commands that occur later in `submission order`. The second synchronization scope is limited to operations on the pipeline stages determined by the `destination stage mask` specified by `dstStageMask`.

The first `access scope` is limited to accesses in the pipeline stages determined by the `source stage mask` specified by `srcStageMask`. Within that, the first access scope only includes the first access scopes defined by elements of the `pMemoryBarriers`, `pBufferMemoryBarriers` and `pImageMemoryBarriers` arrays, which each define a set of `memory barriers`. If no memory barriers are specified, then the first access scope includes no accesses.

The second `access scope` is limited to accesses in the pipeline stages determined by the `destination stage mask` specified by `dstStageMask`. Within that, the second access scope only includes the second access scopes defined by elements of the `pMemoryBarriers`, `pBufferMemoryBarriers` and `pImageMemoryBarriers` arrays, which each define a set of `memory barriers`. If no memory barriers are specified, then the second access scope includes no accesses.

Valid Usage

- VUID-vkCmdWaitEvents-srcStageMask-04090
If the `geometryShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-vkCmdWaitEvents-srcStageMask-04091
If the `tessellationShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdWaitEvents-srcStageMask-07319
If the `attachmentFragmentShadingRate` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdWaitEvents-srcStageMask-03937
If the `synchronization2` feature is not enabled, `srcStageMask` **must** not be 0
- VUID-vkCmdWaitEvents-dstStageMask-04090
If the `geometryShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-vkCmdWaitEvents-dstStageMask-04091
If the `tessellationShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdWaitEvents-dstStageMask-07319
If the `attachmentFragmentShadingRate` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdWaitEvents-dstStageMask-03937
If the `synchronization2` feature is not enabled, `dstStageMask` **must** not be 0

- VUID-vkCmdWaitEvents-srcAccessMask-02815
The `srcAccessMask` member of each element of `pMemoryBarriers` **must** only include access flags that are supported by one or more of the pipeline stages in `srcStageMask`, as specified in the [table of supported access types](#)
- VUID-vkCmdWaitEvents-dstAccessMask-02816
The `dstAccessMask` member of each element of `pMemoryBarriers` **must** only include access flags that are supported by one or more of the pipeline stages in `dstStageMask`, as specified in the [table of supported access types](#)
- VUID-vkCmdWaitEvents-pBufferMemoryBarriers-02817
For any element of `pBufferMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `srcQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `srcAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `srcStageMask`, as specified in the [table of supported access types](#)
- VUID-vkCmdWaitEvents-pBufferMemoryBarriers-02818
For any element of `pBufferMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `dstQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `dstAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `dstStageMask`, as specified in the [table of supported access types](#)
- VUID-vkCmdWaitEvents-pImageMemoryBarriers-02819
For any element of `pImageMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `srcQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `srcAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `srcStageMask`, as specified in the [table of supported access types](#)
- VUID-vkCmdWaitEvents-pImageMemoryBarriers-02820
For any element of `pImageMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `dstQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `dstAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `dstStageMask`, as specified in the [table of supported access types](#)
- VUID-vkCmdWaitEvents-srcStageMask-06459
Any pipeline stage included in `srcStageMask` **must** be supported by the capabilities of the queue family specified by the `queueFamilyIndex` member of the `VkCommandPoolCreateInfo` structure that was used to create the `VkCommandPool` that `commandBuffer` was allocated from, as specified in the [table of supported pipeline stages](#)
- VUID-vkCmdWaitEvents-dstStageMask-06460
Any pipeline stage included in `dstStageMask` **must** be supported by the capabilities of the queue family specified by the `queueFamilyIndex` member of the `VkCommandPoolCreateInfo` structure that was used to create the `VkCommandPool` that

`commandBuffer` was allocated from, as specified in the [table of supported pipeline stages](#)

- VUID-vkCmdWaitEvents-srcStageMask-01158
`srcStageMask` **must** be the bitwise OR of the `stageMask` parameter used in previous calls to `vkCmdSetEvent` with any of the elements of `pEvents` and `VK_PIPELINE_STAGE_HOST_BIT` if any of the elements of `pEvents` was set using `vkSetEvent`
- VUID-vkCmdWaitEvents-srcStageMask-07308
If this command is called inside a render pass instance, `srcStageMask` **must** not include `VK_PIPELINE_STAGE_HOST_BIT`
- VUID-vkCmdWaitEvents-srcQueueFamilyIndex-02803
The `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members of any element of `pBufferMemoryBarriers` or `pImageMemoryBarriers` **must** be equal
- VUID-vkCmdWaitEvents-commandBuffer-01167
`commandBuffer`'s current device mask **must** include exactly one physical device
- VUID-vkCmdWaitEvents-pEvents-03847
Elements of `pEvents` **must** not have been signaled by `vkCmdSetEvent2KHR`

Valid Usage (Implicit)

- VUID-vkCmdWaitEvents-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdWaitEvents-pEvents-parameter
`pEvents` **must** be a valid pointer to an array of `eventCount` valid `VkEvent` handles
- VUID-vkCmdWaitEvents-srcStageMask-parameter
`srcStageMask` **must** be a valid combination of `VkPipelineStageFlagBits` values
- VUID-vkCmdWaitEvents-dstStageMask-parameter
`dstStageMask` **must** be a valid combination of `VkPipelineStageFlagBits` values
- VUID-vkCmdWaitEvents-pMemoryBarriers-parameter
If `memoryBarrierCount` is not 0, `pMemoryBarriers` **must** be a valid pointer to an array of `memoryBarrierCount` valid `VkMemoryBarrier` structures
- VUID-vkCmdWaitEvents-pBufferMemoryBarriers-parameter
If `bufferMemoryBarrierCount` is not 0, `pBufferMemoryBarriers` **must** be a valid pointer to an array of `bufferMemoryBarrierCount` valid `VkBufferMemoryBarrier` structures
- VUID-vkCmdWaitEvents-pImageMemoryBarriers-parameter
If `imageMemoryBarrierCount` is not 0, `pImageMemoryBarriers` **must** be a valid pointer to an array of `imageMemoryBarrierCount` valid `VkImageMemoryBarrier` structures
- VUID-vkCmdWaitEvents-commandBuffer-recording
`commandBuffer` **must** be in the [recording state](#)
- VUID-vkCmdWaitEvents-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdWaitEvents-eventCount-arraylength

`eventCount` **must** be greater than 0

- `VUID-vkCmdWaitEvents-commonparent`

Both of `commandBuffer`, and the elements of `pEvents` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT	Synchronization

7.6. Pipeline Barriers

To record a pipeline barrier, call:

```
// Provided by VK_KHR_synchronization2
void vkCmdPipelineBarrier2KHR(
    VkCommandBuffer                commandBuffer,
    const VkDependencyInfo*        pDependencyInfo);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `pDependencyInfo` is a pointer to a `VkDependencyInfo` structure defining the scopes of this operation.

When `vkCmdPipelineBarrier2KHR` is submitted to a queue, it defines memory dependencies between commands that were submitted to the same queue before it, and those submitted to the same queue after it.

The first `synchronization scope` and `access scope` of each memory dependency defined by `pDependencyInfo` are applied to operations that occurred earlier in `submission order`.

The second `synchronization scope` and `access scope` of each memory dependency defined by `pDependencyInfo` are applied to operations that occurred later in `submission order`.

If `vkCmdPipelineBarrier2KHR` is recorded within a render pass instance, the synchronization scopes are limited to a subset of operations within the same subpass or render pass instance.

Valid Usage

- VUID-vkCmdPipelineBarrier2-None-07889

If `vkCmdPipelineBarrier2KHR` is called within a render pass instance using a `VkRenderPass` object, the render pass **must** have been created with at least one subpass dependency that expresses a dependency from the current subpass to itself, does not include `VK_DEPENDENCY_BY_REGION_BIT` if this command does not, does not include `VK_DEPENDENCY_VIEW_LOCAL_BIT` if this command does not, and has [synchronization scopes](#) and [access scopes](#) that are all supersets of the scopes defined in this command

- VUID-vkCmdPipelineBarrier2-bufferMemoryBarrierCount-01178

If `vkCmdPipelineBarrier2KHR` is called within a render pass instance using a `VkRenderPass` object, it **must** not include any buffer memory barriers

- VUID-vkCmdPipelineBarrier2-image-04073

If `vkCmdPipelineBarrier2KHR` is called within a render pass instance using a `VkRenderPass` object, the `image` member of any image memory barrier included in this command **must** be an attachment used in the current subpass both as an input attachment, and as either a color, or depth/stencil attachment

- VUID-vkCmdPipelineBarrier2-oldLayout-01181

If `vkCmdPipelineBarrier2KHR` is called within a render pass instance, the `oldLayout` and `newLayout` members of any image memory barrier included in this command **must** be equal

- VUID-vkCmdPipelineBarrier2-srcQueueFamilyIndex-01182

If `vkCmdPipelineBarrier2KHR` is called within a render pass instance, the `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members of any memory barrier included in this command **must** be equal

- VUID-vkCmdPipelineBarrier2-None-07890

If `vkCmdPipelineBarrier2KHR` is called within a render pass instance, and the source stage masks of any memory barriers include [framebuffer-space stages](#), destination stage masks of all memory barriers **must** only include [framebuffer-space stages](#)

- VUID-vkCmdPipelineBarrier2-dependencyFlags-07891

If `vkCmdPipelineBarrier2KHR` is called within a render pass instance, and the source stage masks of any memory barriers include [framebuffer-space stages](#), then `dependencyFlags` **must** include `VK_DEPENDENCY_BY_REGION_BIT`

- VUID-vkCmdPipelineBarrier2-None-07892

If `vkCmdPipelineBarrier2KHR` is called within a render pass instance, the source and destination stage masks of any memory barriers **must** only include graphics pipeline stages

- VUID-vkCmdPipelineBarrier2-dependencyFlags-01186

If `vkCmdPipelineBarrier2KHR` is called outside of a render pass instance, the dependency flags **must** not include `VK_DEPENDENCY_VIEW_LOCAL_BIT`

- VUID-vkCmdPipelineBarrier2-None-07893

If `vkCmdPipelineBarrier2KHR` is called inside a render pass instance, and there is more than one view in the current subpass, dependency flags **must** include `VK_DEPENDENCY_VIEW_LOCAL_BIT`

- VUID-vkCmdPipelineBarrier2-synchronization2-03848
The `synchronization2` feature **must** be enabled
- VUID-vkCmdPipelineBarrier2-srcStageMask-09673
The `srcStageMask` member of any element of the `pMemoryBarriers` member of `pDependencyInfo` **must** only include pipeline stages valid for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdPipelineBarrier2-dstStageMask-09674
The `dstStageMask` member of any element of the `pMemoryBarriers` member of `pDependencyInfo` **must** only include pipeline stages valid for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdPipelineBarrier2-srcStageMask-09675
If a buffer or image memory barrier does not specify an `acquire operation`, the respective `srcStageMask` member of the element of the `pBufferMemoryBarriers` or `pImageMemoryBarriers` members of `pDependencyInfo` **must** only include pipeline stages valid for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdPipelineBarrier2-dstStageMask-09676
If a buffer or image memory barrier does not specify an `release operation`, the respective `dstStageMask` member of the element of the `pBufferMemoryBarriers` or `pImageMemoryBarriers` members of `pDependencyInfo` **must** only include pipeline stages valid for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdPipelineBarrier2-srcQueueFamilyIndex-10387
If a buffer or image memory barrier specifies a `queue family ownership transfer operation`, either the `srcQueueFamilyIndex` or `dstQueueFamilyIndex` member of the element of the `pBufferMemoryBarriers` or `pImageMemoryBarriers` members of `pDependencyInfo` and the queue family index that was used to create the command pool that `commandBuffer` was allocated from **must** be equal

Valid Usage (Implicit)

- VUID-vkCmdPipelineBarrier2-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdPipelineBarrier2-pDependencyInfo-parameter
`pDependencyInfo` **must** be a valid pointer to a valid `VkDependencyInfo` structure
- VUID-vkCmdPipelineBarrier2-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdPipelineBarrier2-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Synchronization

To record a pipeline barrier, call:

```
// Provided by VK_VERSION_1_0
void vkCmdPipelineBarrier(
    VkCommandBuffer          commandBuffer,
    VkPipelineStageFlags     srcStageMask,
    VkPipelineStageFlags     dstStageMask,
    VkDependencyFlags        dependencyFlags,
    uint32_t                 memoryBarrierCount,
    const VkMemoryBarrier*   pMemoryBarriers,
    uint32_t                 bufferMemoryBarrierCount,
    const VkBufferMemoryBarrier* pBufferMemoryBarriers,
    uint32_t                 imageMemoryBarrierCount,
    const VkImageMemoryBarrier* pImageMemoryBarriers);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `srcStageMask` is a bitmask of `VkPipelineStageFlagBits` specifying the [source stages](#).
- `dstStageMask` is a bitmask of `VkPipelineStageFlagBits` specifying the [destination stages](#).
- `dependencyFlags` is a bitmask of `VkDependencyFlagBits` specifying how execution and memory dependencies are formed.
- `memoryBarrierCount` is the length of the `pMemoryBarriers` array.
- `pMemoryBarriers` is a pointer to an array of `VkMemoryBarrier` structures.
- `bufferMemoryBarrierCount` is the length of the `pBufferMemoryBarriers` array.
- `pBufferMemoryBarriers` is a pointer to an array of `VkBufferMemoryBarrier` structures.

- `imageMemoryBarrierCount` is the length of the `pImageMemoryBarriers` array.
- `pImageMemoryBarriers` is a pointer to an array of `VkImageMemoryBarrier` structures.

`vkCmdPipelineBarrier` operates almost identically to `vkCmdPipelineBarrier2KHR`, except that the scopes and barriers are defined as direct parameters rather than being defined by a `VkDependencyInfo`.

When `vkCmdPipelineBarrier` is submitted to a queue, it defines a memory dependency between commands that were submitted to the same queue before it, and those submitted to the same queue after it.

If `vkCmdPipelineBarrier` was recorded outside a render pass instance, the first `synchronization scope` includes all commands that occur earlier in `submission order`. If `vkCmdPipelineBarrier` was recorded inside a render pass instance, the first synchronization scope includes only commands that occur earlier in `submission order` within the same subpass. In either case, the first synchronization scope is limited to operations on the pipeline stages determined by the `source stage mask` specified by `srcStageMask`.

If `vkCmdPipelineBarrier` was recorded outside a render pass instance, the second `synchronization scope` includes all commands that occur later in `submission order`. If `vkCmdPipelineBarrier` was recorded inside a render pass instance, the second synchronization scope includes only commands that occur later in `submission order` within the same subpass. In either case, the second synchronization scope is limited to operations on the pipeline stages determined by the `destination stage mask` specified by `dstStageMask`.

The first `access scope` is limited to accesses in the pipeline stages determined by the `source stage mask` specified by `srcStageMask`. Within that, the first access scope only includes the first access scopes defined by elements of the `pMemoryBarriers`, `pBufferMemoryBarriers` and `pImageMemoryBarriers` arrays, which each define a set of `memory barriers`. If no memory barriers are specified, then the first access scope includes no accesses.

The second `access scope` is limited to accesses in the pipeline stages determined by the `destination stage mask` specified by `dstStageMask`. Within that, the second access scope only includes the second access scopes defined by elements of the `pMemoryBarriers`, `pBufferMemoryBarriers` and `pImageMemoryBarriers` arrays, which each define a set of `memory barriers`. If no memory barriers are specified, then the second access scope includes no accesses.

If `dependencyFlags` includes `VK_DEPENDENCY_BY_REGION_BIT`, then any dependency between `framebuffer-space` pipeline stages is `framebuffer-local` - otherwise it is `framebuffer-global`.

Valid Usage

- VUID-vkCmdPipelineBarrier-srcStageMask-04090
If the `geometryShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-vkCmdPipelineBarrier-srcStageMask-04091
If the `tessellationShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or

VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT

- VUID-vkCmdPipelineBarrier-srcStageMask-07319
If the `attachmentFragmentShadingRate` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdPipelineBarrier-srcStageMask-03937
If the `synchronization2` feature is not enabled, `srcStageMask` **must** not be 0
- VUID-vkCmdPipelineBarrier-dstStageMask-04090
If the `geometryShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-vkCmdPipelineBarrier-dstStageMask-04091
If the `tessellationShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdPipelineBarrier-dstStageMask-07319
If the `attachmentFragmentShadingRate` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdPipelineBarrier-dstStageMask-03937
If the `synchronization2` feature is not enabled, `dstStageMask` **must** not be 0
- VUID-vkCmdPipelineBarrier-srcAccessMask-02815
The `srcAccessMask` member of each element of `pMemoryBarriers` **must** only include access flags that are supported by one or more of the pipeline stages in `srcStageMask`, as specified in the [table of supported access types](#)
- VUID-vkCmdPipelineBarrier-dstAccessMask-02816
The `dstAccessMask` member of each element of `pMemoryBarriers` **must** only include access flags that are supported by one or more of the pipeline stages in `dstStageMask`, as specified in the [table of supported access types](#)
- VUID-vkCmdPipelineBarrier-pBufferMemoryBarriers-02817
For any element of `pBufferMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `srcQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `srcAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `srcStageMask`, as specified in the [table of supported access types](#)
- VUID-vkCmdPipelineBarrier-pBufferMemoryBarriers-02818
For any element of `pBufferMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `dstQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `dstAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `dstStageMask`, as specified in the [table of supported access types](#)
- VUID-vkCmdPipelineBarrier-pImageMemoryBarriers-02819
For any element of `pImageMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `srcQueueFamilyIndex` is the queue family

index that was used to create the command pool that `commandBuffer` was allocated from, then its `srcAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `srcStageMask`, as specified in the [table of supported access types](#)

- VUID-vkCmdPipelineBarrier-pImageMemoryBarriers-02820

For any element of `pImageMemoryBarriers`, if its `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members are equal, or if its `dstQueueFamilyIndex` is the queue family index that was used to create the command pool that `commandBuffer` was allocated from, then its `dstAccessMask` member **must** only contain access flags that are supported by one or more of the pipeline stages in `dstStageMask`, as specified in the [table of supported access types](#)

- VUID-vkCmdPipelineBarrier-None-07889

If `vkCmdPipelineBarrier` is called within a render pass instance using a `VkRenderPass` object, the render pass **must** have been created with at least one subpass dependency that expresses a dependency from the current subpass to itself, does not include `VK_DEPENDENCY_BY_REGION_BIT` if this command does not, does not include `VK_DEPENDENCY_VIEW_LOCAL_BIT` if this command does not, and has [synchronization scopes](#) and [access scopes](#) that are all supersets of the scopes defined in this command

- VUID-vkCmdPipelineBarrier-bufferMemoryBarrierCount-01178

If `vkCmdPipelineBarrier` is called within a render pass instance using a `VkRenderPass` object, it **must** not include any buffer memory barriers

- VUID-vkCmdPipelineBarrier-image-04073

If `vkCmdPipelineBarrier` is called within a render pass instance using a `VkRenderPass` object, the `image` member of any image memory barrier included in this command **must** be an attachment used in the current subpass both as an input attachment, and as either a color, or depth/stencil attachment

- VUID-vkCmdPipelineBarrier-oldLayout-01181

If `vkCmdPipelineBarrier` is called within a render pass instance, the `oldLayout` and `newLayout` members of any image memory barrier included in this command **must** be equal

- VUID-vkCmdPipelineBarrier-srcQueueFamilyIndex-01182

If `vkCmdPipelineBarrier` is called within a render pass instance, the `srcQueueFamilyIndex` and `dstQueueFamilyIndex` members of any memory barrier included in this command **must** be equal

- VUID-vkCmdPipelineBarrier-None-07890

If `vkCmdPipelineBarrier` is called within a render pass instance, and the source stage masks of any memory barriers include [framebuffer-space stages](#), destination stage masks of all memory barriers **must** only include [framebuffer-space stages](#)

- VUID-vkCmdPipelineBarrier-dependencyFlags-07891

If `vkCmdPipelineBarrier` is called within a render pass instance, and the source stage masks of any memory barriers include [framebuffer-space stages](#), then `dependencyFlags` **must** include `VK_DEPENDENCY_BY_REGION_BIT`

- VUID-vkCmdPipelineBarrier-None-07892

If `vkCmdPipelineBarrier` is called within a render pass instance, the source and destination

stage masks of any memory barriers **must** only include graphics pipeline stages

- VUID-vkCmdPipelineBarrier-dependencyFlags-01186

If `vkCmdPipelineBarrier` is called outside of a render pass instance, the dependency flags **must** not include `VK_DEPENDENCY_VIEW_LOCAL_BIT`

- VUID-vkCmdPipelineBarrier-None-07893

If `vkCmdPipelineBarrier` is called inside a render pass instance, and there is more than one view in the current subpass, dependency flags **must** include `VK_DEPENDENCY_VIEW_LOCAL_BIT`

- VUID-vkCmdPipelineBarrier-oldLayout-10758

If called within a render pass instance using a `VkRenderPass` object, the `oldLayout` member of any image memory barrier included in this command **must** be equal to the layout that the corresponding attachment uses during the subpass

- VUID-vkCmdPipelineBarrier-srcStageMask-06461

Any pipeline stage included in `srcStageMask` **must** be supported by the capabilities of the queue family specified by the `queueFamilyIndex` member of the `VkCommandPoolCreateInfo` structure that was used to create the `VkCommandPool` that `commandBuffer` was allocated from, as specified in the [table of supported pipeline stages](#)

- VUID-vkCmdPipelineBarrier-dstStageMask-06462

Any pipeline stage included in `dstStageMask` **must** be supported by the capabilities of the queue family specified by the `queueFamilyIndex` member of the `VkCommandPoolCreateInfo` structure that was used to create the `VkCommandPool` that `commandBuffer` was allocated from, as specified in the [table of supported pipeline stages](#)

- VUID-vkCmdPipelineBarrier-srcStageMask-09633

If either `srcStageMask` or `dstStageMask` includes `VK_PIPELINE_STAGE_HOST_BIT`, for any element of `pImageMemoryBarriers`, `srcQueueFamilyIndex` and `dstQueueFamilyIndex` **must** be equal

- VUID-vkCmdPipelineBarrier-srcStageMask-09634

If either `srcStageMask` or `dstStageMask` includes `VK_PIPELINE_STAGE_HOST_BIT`, for any element of `pBufferMemoryBarriers`, `srcQueueFamilyIndex` and `dstQueueFamilyIndex` **must** be equal

- VUID-vkCmdPipelineBarrier-srcQueueFamilyIndex-10388

If a buffer or image memory barrier specifies a [queue family ownership transfer operation](#), either the `srcQueueFamilyIndex` or `dstQueueFamilyIndex` member and the queue family index that was used to create the command pool that `commandBuffer` was allocated from **must** be equal

Valid Usage (Implicit)

- VUID-vkCmdPipelineBarrier-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle

- VUID-vkCmdPipelineBarrier-srcStageMask-parameter
`srcStageMask` **must** be a valid combination of `VkPipelineStageFlagBits` values

- VUID-vkCmdPipelineBarrier-dstStageMask-parameter
`dstStageMask` **must** be a valid combination of `VkPipelineStageFlagBits` values

- VUID-vkCmdPipelineBarrier-dependencyFlags-parameter
`dependencyFlags` **must** be a valid combination of `VkDependencyFlagBits` values
- VUID-vkCmdPipelineBarrier-pMemoryBarriers-parameter
If `memoryBarrierCount` is not 0, `pMemoryBarriers` **must** be a valid pointer to an array of `memoryBarrierCount` valid `VkMemoryBarrier` structures
- VUID-vkCmdPipelineBarrier-pBufferMemoryBarriers-parameter
If `bufferMemoryBarrierCount` is not 0, `pBufferMemoryBarriers` **must** be a valid pointer to an array of `bufferMemoryBarrierCount` valid `VkBufferMemoryBarrier` structures
- VUID-vkCmdPipelineBarrier-pImageMemoryBarriers-parameter
If `imageMemoryBarrierCount` is not 0, `pImageMemoryBarriers` **must** be a valid pointer to an array of `imageMemoryBarrierCount` valid `VkImageMemoryBarrier` structures
- VUID-vkCmdPipelineBarrier-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdPipelineBarrier-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Synchronization

Bits which **can** be set in `vkCmdPipelineBarrier::dependencyFlags`, specifying how execution and memory dependencies are formed, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkDependencyFlagBits {
    VK_DEPENDENCY_BY_REGION_BIT = 0x00000001,
    // Provided by VK_VERSION_1_1
    VK_DEPENDENCY_DEVICE_GROUP_BIT = 0x00000004,
    // Provided by VK_VERSION_1_1
    VK_DEPENDENCY_VIEW_LOCAL_BIT = 0x00000002,
} VkDependencyFlagBits;
```

- **VK_DEPENDENCY_BY_REGION_BIT** specifies that dependencies will be [framebuffer-local](#).
- **VK_DEPENDENCY_VIEW_LOCAL_BIT** specifies that dependencies will be [view-local](#).
- **VK_DEPENDENCY_DEVICE_GROUP_BIT** specifies that dependencies are [non-device-local](#).

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkDependencyFlags;
```

VkDependencyFlags is a bitmask type for setting a mask of zero or more [VkDependencyFlagBits](#).

7.7. Memory Barriers

Memory barriers are used to explicitly control access to buffer and image subresource ranges. Memory barriers are used to [transfer ownership between queue families](#), [change image layouts](#), and define [availability and visibility operations](#). They explicitly define the [access types](#) and buffer and image subresource ranges that are included in the [access scopes](#) of a memory dependency that is created by a synchronization command that includes them.

7.7.1. Global Memory Barriers

Global memory barriers apply to memory accesses involving all memory objects that exist at the time of its execution.

The **VkMemoryBarrier2** structure is defined as:

```
typedef struct VkMemoryBarrier2 {
    VkStructureType      sType;
    const void*          pNext;
    VkPipelineStageFlags2 srcStageMask;
    VkAccessFlags2       srcAccessMask;
    VkPipelineStageFlags2 dstStageMask;
    VkAccessFlags2       dstAccessMask;
} VkMemoryBarrier2;
```

```
// Provided by VK_KHR_synchronization2
typedef VkMemoryBarrier2 VkMemoryBarrier2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcStageMask` is a `VkPipelineStageFlags2` mask of pipeline stages to be included in the [first synchronization scope](#).
- `srcAccessMask` is a `VkAccessFlags2` mask of access flags to be included in the [first access scope](#).
- `dstStageMask` is a `VkPipelineStageFlags2` mask of pipeline stages to be included in the [second synchronization scope](#).
- `dstAccessMask` is a `VkAccessFlags2` mask of access flags to be included in the [second access scope](#).

This structure defines a [memory dependency](#) affecting all device memory.

The first [synchronization scope](#) and [access scope](#) described by this structure include only operations and memory accesses specified by the source stage mask and source access mask.

The second [synchronization scope](#) and [access scope](#) described by this structure include only operations and memory accesses specified by destination stage mask and destination access mask.

Valid Usage

- VUID-VkMemoryBarrier2-srcStageMask-03929
If the `geometryShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`
- VUID-VkMemoryBarrier2-srcStageMask-03930
If the `tessellationShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-VkMemoryBarrier2-srcStageMask-07317
If the `attachmentFragmentShadingRate` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-VkMemoryBarrier2-srcAccessMask-03900
If `srcAccessMask` includes `VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkMemoryBarrier2-srcAccessMask-03901
If `srcAccessMask` includes `VK_ACCESS_2_INDEX_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkMemoryBarrier2-srcAccessMask-03902
If `srcAccessMask` includes `VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT`, `VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkMemoryBarrier2-srcAccessMask-03903

If `srcAccessMask` includes `VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT`, `VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-srcAccessMask-03904

If `srcAccessMask` includes `VK_ACCESS_2_UNIFORM_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkMemoryBarrier2-srcAccessMask-03905

If `srcAccessMask` includes `VK_ACCESS_2_SHADER_SAMPLED_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkMemoryBarrier2-srcAccessMask-03906

If `srcAccessMask` includes `VK_ACCESS_2_SHADER_STORAGE_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkMemoryBarrier2-srcAccessMask-03907

If `srcAccessMask` includes `VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkMemoryBarrier2-srcAccessMask-07454

If `srcAccessMask` includes `VK_ACCESS_2_SHADER_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkMemoryBarrier2-srcAccessMask-03909

If `srcAccessMask` includes `VK_ACCESS_2_SHADER_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkMemoryBarrier2-srcAccessMask-03910

If `srcAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-srcAccessMask-03911

If `srcAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-srcAccessMask-03912

If `srcAccessMask` includes `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-srcAccessMask-03913

If `srcAccessMask` includes `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`,

VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkMemoryBarrier2-srcAccessMask-03914

If `srcAccessMask` includes `VK_ACCESS_2_TRANSFER_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COPY_BIT`, `VK_PIPELINE_STAGE_2_BLIT_BIT`, `VK_PIPELINE_STAGE_2_RESOLVE_BIT`, `VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-srcAccessMask-03915

If `srcAccessMask` includes `VK_ACCESS_2_TRANSFER_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COPY_BIT`, `VK_PIPELINE_STAGE_2_BLIT_BIT`, `VK_PIPELINE_STAGE_2_RESOLVE_BIT`, `VK_PIPELINE_STAGE_2_CLEAR_BIT`, `VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-srcAccessMask-03916

If `srcAccessMask` includes `VK_ACCESS_2_HOST_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_HOST_BIT`

- VUID-VkMemoryBarrier2-srcAccessMask-03917

If `srcAccessMask` includes `VK_ACCESS_2_HOST_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_HOST_BIT`

- VUID-VkMemoryBarrier2-srcAccessMask-03926

If `srcAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-dstStageMask-03929

If the `geometryShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`

- VUID-VkMemoryBarrier2-dstStageMask-03930

If the `tessellationShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`

- VUID-VkMemoryBarrier2-dstStageMask-07317

If the `attachmentFragmentShadingRate` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`

- VUID-VkMemoryBarrier2-dstAccessMask-03900

If `dstAccessMask` includes `VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-dstAccessMask-03901

If `dstAccessMask` includes `VK_ACCESS_2_INDEX_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-dstAccessMask-03902

If `dstAccessMask` includes `VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT`,

VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or
VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkMemoryBarrier2-dstAccessMask-03903

If `dstAccessMask` includes `VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT`, `VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-dstAccessMask-03904

If `dstAccessMask` includes `VK_ACCESS_2_UNIFORM_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkMemoryBarrier2-dstAccessMask-03905

If `dstAccessMask` includes `VK_ACCESS_2_SHADER_SAMPLED_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkMemoryBarrier2-dstAccessMask-03906

If `dstAccessMask` includes `VK_ACCESS_2_SHADER_STORAGE_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkMemoryBarrier2-dstAccessMask-03907

If `dstAccessMask` includes `VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkMemoryBarrier2-dstAccessMask-07454

If `dstAccessMask` includes `VK_ACCESS_2_SHADER_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkMemoryBarrier2-dstAccessMask-03909

If `dstAccessMask` includes `VK_ACCESS_2_SHADER_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkMemoryBarrier2-dstAccessMask-03910

If `dstAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-dstAccessMask-03911

If `dstAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-dstAccessMask-03912

If `dstAccessMask` includes `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkMemoryBarrier2-dstAccessMask-03913
If `dstAccessMask` includes `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkMemoryBarrier2-dstAccessMask-03914
If `dstAccessMask` includes `VK_ACCESS_2_TRANSFER_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_COPY_BIT`, `VK_PIPELINE_STAGE_2_BLIT_BIT`, `VK_PIPELINE_STAGE_2_RESOLVE_BIT`, `VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkMemoryBarrier2-dstAccessMask-03915
If `dstAccessMask` includes `VK_ACCESS_2_TRANSFER_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_COPY_BIT`, `VK_PIPELINE_STAGE_2_BLIT_BIT`, `VK_PIPELINE_STAGE_2_RESOLVE_BIT`, `VK_PIPELINE_STAGE_2_CLEAR_BIT`, `VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkMemoryBarrier2-dstAccessMask-03916
If `dstAccessMask` includes `VK_ACCESS_2_HOST_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_HOST_BIT`
- VUID-VkMemoryBarrier2-dstAccessMask-03917
If `dstAccessMask` includes `VK_ACCESS_2_HOST_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_HOST_BIT`
- VUID-VkMemoryBarrier2-dstAccessMask-03926
If `dstAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

Valid Usage (Implicit)

- VUID-VkMemoryBarrier2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_BARRIER_2`
- VUID-VkMemoryBarrier2-srcStageMask-parameter
`srcStageMask` **must** be a valid combination of `VkPipelineStageFlagBits2` values
- VUID-VkMemoryBarrier2-srcAccessMask-parameter
`srcAccessMask` **must** be a valid combination of `VkAccessFlagBits2` values
- VUID-VkMemoryBarrier2-dstStageMask-parameter
`dstStageMask` **must** be a valid combination of `VkPipelineStageFlagBits2` values
- VUID-VkMemoryBarrier2-dstAccessMask-parameter
`dstAccessMask` **must** be a valid combination of `VkAccessFlagBits2` values

The `VkMemoryBarrier` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkMemoryBarrier {
    VkStructureType    sType;
    const void*        pNext;
    VkAccessFlags       srcAccessMask;
    VkAccessFlags       dstAccessMask;
} VkMemoryBarrier;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **srcAccessMask** is a bitmask of [VkAccessFlagBits](#) specifying a [source access mask](#).
- **dstAccessMask** is a bitmask of [VkAccessFlagBits](#) specifying a [destination access mask](#).

The first [access scope](#) is limited to access types in the [source access mask](#) specified by **srcAccessMask**.

The second [access scope](#) is limited to access types in the [destination access mask](#) specified by **dstAccessMask**.

Valid Usage (Implicit)

- VUID-VkMemoryBarrier-sType-sType
sType must be [VK_STRUCTURE_TYPE_MEMORY_BARRIER](#)
- VUID-VkMemoryBarrier-pNext-pNext
pNext must be **NULL**
- VUID-VkMemoryBarrier-srcAccessMask-parameter
srcAccessMask must be a valid combination of [VkAccessFlagBits](#) values
- VUID-VkMemoryBarrier-dstAccessMask-parameter
dstAccessMask must be a valid combination of [VkAccessFlagBits](#) values

7.7.2. Buffer Memory Barriers

Buffer memory barriers only apply to memory accesses involving a specific buffer range. That is, a memory dependency formed from a buffer memory barrier is [scoped](#) to access via the specified buffer range. Buffer memory barriers **can** also be used to define a [queue family ownership transfer](#) for the specified buffer range.

The [VkBufferMemoryBarrier2](#) structure is defined as:

```
typedef struct VkBufferMemoryBarrier2 {
    VkStructureType      sType;
    const void*          pNext;
    VkPipelineStageFlags2 srcStageMask;
    VkAccessFlags2       srcAccessMask;
    VkPipelineStageFlags2 dstStageMask;
    VkAccessFlags2       dstAccessMask;
    uint32_t             srcQueueFamilyIndex;
    uint32_t             dstQueueFamilyIndex;
    VkBuffer              buffer;
    VkDeviceSize          offset;
    VkDeviceSize          size;
} VkBufferMemoryBarrier2;
```

```
// Provided by VK_KHR_synchronization2
typedef VkBufferMemoryBarrier2 VkBufferMemoryBarrier2KHR;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is `NULL` or a pointer to a structure extending this structure.
- **srcStageMask** is a [VkPipelineStageFlags2](#) mask of pipeline stages to be included in the [first synchronization scope](#).
- **srcAccessMask** is a [VkAccessFlags2](#) mask of access flags to be included in the [first access scope](#).
- **dstStageMask** is a [VkPipelineStageFlags2](#) mask of pipeline stages to be included in the [second synchronization scope](#).
- **dstAccessMask** is a [VkAccessFlags2](#) mask of access flags to be included in the [second access scope](#).
- **srcQueueFamilyIndex** is the source queue family for a [queue family ownership transfer](#).
- **dstQueueFamilyIndex** is the destination queue family for a [queue family ownership transfer](#).
- **buffer** is a handle to the buffer whose backing memory is affected by the barrier.
- **offset** is an offset in bytes into the backing memory for **buffer**; this is relative to the base offset as bound to the buffer (see [vkBindBufferMemory](#)).
- **size** is a size in bytes of the affected area of backing memory for **buffer**, or `VK_WHOLE_SIZE` to use the range from **offset** to the end of the buffer.

This structure defines a [memory dependency](#) limited to a range of a buffer, and **can** define a [queue family ownership transfer operation](#) for that range.

The first [synchronization scope](#) and [access scope](#) described by this structure include only operations and memory accesses specified by the source stage mask and the source access mask.

The second [synchronization scope](#) and [access scope](#) described by this structure include only operations and memory accesses specified by the destination stage mask and the destination access mask.

Both [access scopes](#) are limited to only memory accesses to [buffer](#) in the range defined by [offset](#) and [size](#).

If [buffer](#) was created with [VK_SHARING_MODE_EXCLUSIVE](#), and [srcQueueFamilyIndex](#) is not equal to [dstQueueFamilyIndex](#), this memory barrier defines a [queue family ownership transfer operation](#). When executed on a queue in the family identified by [srcQueueFamilyIndex](#), this barrier defines a [queue family release operation](#) for the specified buffer range, and the second synchronization scope does not apply to this operation. When executed on a queue in the family identified by [dstQueueFamilyIndex](#), this barrier defines a [queue family acquire operation](#) for the specified buffer range, and the first synchronization scope does not apply to this operation.

A [queue family ownership transfer operation](#) is also defined if the values are not equal, and either is one of the special queue family values reserved for external memory ownership transfers, as described in [Queue Family Ownership Transfer](#). A [queue family release operation](#) is defined when [dstQueueFamilyIndex](#) is one of those values, and a [queue family acquire operation](#) is defined when [srcQueueFamilyIndex](#) is one of those values.

Valid Usage

- VUID-VkBufferMemoryBarrier2-srcStageMask-03929
If the [geometryShader](#) feature is not enabled, [srcStageMask](#) **must** not contain [VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT](#)
- VUID-VkBufferMemoryBarrier2-srcStageMask-03930
If the [tessellationShader](#) feature is not enabled, [srcStageMask](#) **must** not contain [VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT](#) or [VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT](#)
- VUID-VkBufferMemoryBarrier2-srcStageMask-07317
If the [attachmentFragmentShadingRate](#) feature is not enabled, [srcStageMask](#) **must** not contain [VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR](#)
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03900
If [srcAccessMask](#) includes [VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT](#), [srcStageMask](#) **must** include [VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT](#), [VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT](#), or [VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT](#)
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03901
If [srcAccessMask](#) includes [VK_ACCESS_2_INDEX_READ_BIT](#), [srcStageMask](#) **must** include [VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT](#), [VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT](#), [VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT](#), or [VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT](#)
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03902
If [srcAccessMask](#) includes [VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT](#), [srcStageMask](#) **must** include [VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT](#), [VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT](#), [VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT](#), or [VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT](#)
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03903
If [srcAccessMask](#) includes [VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT](#), [srcStageMask](#) **must** include [VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT](#),

VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkBufferMemoryBarrier2-srcAccessMask-03904
If `srcAccessMask` includes `VK_ACCESS_2_UNIFORM_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_2_SHADER_BIT` stages
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03905
If `srcAccessMask` includes `VK_ACCESS_2_SHADER_SAMPLED_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_2_SHADER_BIT` stages
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03906
If `srcAccessMask` includes `VK_ACCESS_2_SHADER_STORAGE_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_2_SHADER_BIT` stages
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03907
If `srcAccessMask` includes `VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_2_SHADER_BIT` stages
- VUID-VkBufferMemoryBarrier2-srcAccessMask-07454
If `srcAccessMask` includes `VK_ACCESS_2_SHADER_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_2_SHADER_BIT` stages
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03909
If `srcAccessMask` includes `VK_ACCESS_2_SHADER_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_2_SHADER_BIT` stages
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03910
If `srcAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03911
If `srcAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03912
If `srcAccessMask` includes `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03913
If `srcAccessMask` includes `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkBufferMemoryBarrier2-srcAccessMask-03914
If `srcAccessMask` includes `VK_ACCESS_2_TRANSFER_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COPY_BIT`, `VK_PIPELINE_STAGE_2_BLIT_BIT`, `VK_PIPELINE_STAGE_2_RESOLVE_BIT`, `VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03915
If `srcAccessMask` includes `VK_ACCESS_2_TRANSFER_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COPY_BIT`, `VK_PIPELINE_STAGE_2_BLIT_BIT`, `VK_PIPELINE_STAGE_2_RESOLVE_BIT`, `VK_PIPELINE_STAGE_2_CLEAR_BIT`, `VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03916
If `srcAccessMask` includes `VK_ACCESS_2_HOST_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_HOST_BIT`
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03917
If `srcAccessMask` includes `VK_ACCESS_2_HOST_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_HOST_BIT`
- VUID-VkBufferMemoryBarrier2-srcAccessMask-03926
If `srcAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-dstStageMask-03929
If the `geometryShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`
- VUID-VkBufferMemoryBarrier2-dstStageMask-03930
If the `tessellationShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-VkBufferMemoryBarrier2-dstStageMask-07317
If the `attachmentFragmentShadingRate` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03900
If `dstAccessMask` includes `VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03901
If `dstAccessMask` includes `VK_ACCESS_2_INDEX_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03902
If `dstAccessMask` includes `VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT`, `VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkBufferMemoryBarrier2-dstAccessMask-03903
If `dstAccessMask` includes `VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT`, `VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03904
If `dstAccessMask` includes `VK_ACCESS_2_UNIFORM_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03905
If `dstAccessMask` includes `VK_ACCESS_2_SHADER_SAMPLED_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03906
If `dstAccessMask` includes `VK_ACCESS_2_SHADER_STORAGE_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03907
If `dstAccessMask` includes `VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages
- VUID-VkBufferMemoryBarrier2-dstAccessMask-07454
If `dstAccessMask` includes `VK_ACCESS_2_SHADER_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03909
If `dstAccessMask` includes `VK_ACCESS_2_SHADER_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03910
If `dstAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03911
If `dstAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03912
If `dstAccessMask` includes `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkBufferMemoryBarrier2-dstAccessMask-03913
If `dstAccessMask` includes `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`, `dstStageMask`

must include VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkBufferMemoryBarrier2-dstAccessMask-03914

If `dstAccessMask` includes VK_ACCESS_2_TRANSFER_READ_BIT, `dstStageMask` **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkBufferMemoryBarrier2-dstAccessMask-03915

If `dstAccessMask` includes VK_ACCESS_2_TRANSFER_WRITE_BIT, `dstStageMask` **must** include VK_PIPELINE_STAGE_2_COPY_BIT, VK_PIPELINE_STAGE_2_BLIT_BIT, VK_PIPELINE_STAGE_2_RESOLVE_BIT, VK_PIPELINE_STAGE_2_CLEAR_BIT, VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkBufferMemoryBarrier2-dstAccessMask-03916

If `dstAccessMask` includes VK_ACCESS_2_HOST_READ_BIT, `dstStageMask` **must** include VK_PIPELINE_STAGE_2_HOST_BIT

- VUID-VkBufferMemoryBarrier2-dstAccessMask-03917

If `dstAccessMask` includes VK_ACCESS_2_HOST_WRITE_BIT, `dstStageMask` **must** include VK_PIPELINE_STAGE_2_HOST_BIT

- VUID-VkBufferMemoryBarrier2-dstAccessMask-03926

If `dstAccessMask` includes VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT, `dstStageMask` **must** include VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT, VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT, or VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT

- VUID-VkBufferMemoryBarrier2-offset-01187

`offset` **must** be less than the size of `buffer`

- VUID-VkBufferMemoryBarrier2-size-01188

If `size` is not equal to VK_WHOLE_SIZE, `size` **must** be greater than 0

- VUID-VkBufferMemoryBarrier2-size-01189

If `size` is not equal to VK_WHOLE_SIZE, `size` **must** be less than or equal to the size of `buffer` minus `offset`

- VUID-VkBufferMemoryBarrier2-buffer-01931

If `buffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-VkBufferMemoryBarrier2-buffer-09095

If `buffer` was created with a sharing mode of VK_SHARING_MODE_EXCLUSIVE, and `srcQueueFamilyIndex` and `dstQueueFamilyIndex` are not equal, `srcQueueFamilyIndex` **must** be VK_QUEUE_FAMILY_EXTERNAL, VK_QUEUE_FAMILY_FOREIGN_EXT, or a valid queue family

- VUID-VkBufferMemoryBarrier2-buffer-09096

If `buffer` was created with a sharing mode of VK_SHARING_MODE_EXCLUSIVE, and `srcQueueFamilyIndex` and `dstQueueFamilyIndex` are not equal, `dstQueueFamilyIndex` **must** be VK_QUEUE_FAMILY_EXTERNAL, VK_QUEUE_FAMILY_FOREIGN_EXT, or a valid queue family

- VUID-VkBufferMemoryBarrier2-srcQueueFamilyIndex-09099

If the VK_EXT_queue_family_foreign extension is not enabled `srcQueueFamilyIndex` **must**

not be `VK_QUEUE_FAMILY_FOREIGN_EXT`

- VUID-VkBufferMemoryBarrier2-dstQueueFamilyIndex-09100
If the `VK_EXT_queue_family_foreign` extension is not enabled `dstQueueFamilyIndex` **must** not be `VK_QUEUE_FAMILY_FOREIGN_EXT`
- VUID-VkBufferMemoryBarrier2-srcStageMask-03851
If either `srcStageMask` or `dstStageMask` includes `VK_PIPELINE_STAGE_2_HOST_BIT`, `srcQueueFamilyIndex` and `dstQueueFamilyIndex` **must** be equal

Valid Usage (Implicit)

- VUID-VkBufferMemoryBarrier2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER_2`
- VUID-VkBufferMemoryBarrier2-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkBufferMemoryBarrier2-srcStageMask-parameter
`srcStageMask` **must** be a valid combination of `VkPipelineStageFlagBits2` values
- VUID-VkBufferMemoryBarrier2-srcAccessMask-parameter
`srcAccessMask` **must** be a valid combination of `VkAccessFlagBits2` values
- VUID-VkBufferMemoryBarrier2-dstStageMask-parameter
`dstStageMask` **must** be a valid combination of `VkPipelineStageFlagBits2` values
- VUID-VkBufferMemoryBarrier2-dstAccessMask-parameter
`dstAccessMask` **must** be a valid combination of `VkAccessFlagBits2` values
- VUID-VkBufferMemoryBarrier2-buffer-parameter
`buffer` **must** be a valid `VkBuffer` handle

The `VkBufferMemoryBarrier` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBufferMemoryBarrier {
    VkStructureType    sType;
    const void*        pNext;
    VkAccessFlags       srcAccessMask;
    VkAccessFlags       dstAccessMask;
    uint32_t            srcQueueFamilyIndex;
    uint32_t            dstQueueFamilyIndex;
    VkBuffer            buffer;
    VkDeviceSize        offset;
    VkDeviceSize        size;
} VkBufferMemoryBarrier;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- `srcAccessMask` is a bitmask of `VkAccessFlagBits` specifying a [source access mask](#).
- `dstAccessMask` is a bitmask of `VkAccessFlagBits` specifying a [destination access mask](#).
- `srcQueueFamilyIndex` is the source queue family for a [queue family ownership transfer](#).
- `dstQueueFamilyIndex` is the destination queue family for a [queue family ownership transfer](#).
- `buffer` is a handle to the buffer whose backing memory is affected by the barrier.
- `offset` is an offset in bytes into the backing memory for `buffer`; this is relative to the base offset as bound to the buffer (see [vkBindBufferMemory](#)).
- `size` is a size in bytes of the affected area of backing memory for `buffer`, or `VK_WHOLE_SIZE` to use the range from `offset` to the end of the buffer.

The first [access scope](#) is limited to access to memory through the specified buffer range, via access types in the [source access mask](#) specified by `srcAccessMask`. If the source access mask includes `VK_ACCESS_HOST_WRITE_BIT`, a [memory domain operation](#) is performed where available memory in the host domain is also made available to the device domain.

The second [access scope](#) is limited to access to memory through the specified buffer range, via access types in the [destination access mask](#) specified by `dstAccessMask`. If the destination access mask includes `VK_ACCESS_HOST_WRITE_BIT` or `VK_ACCESS_HOST_READ_BIT`, a [memory domain operation](#) is performed where available memory in the device domain is also made available to the host domain.

NOTE	When <code>VK_MEMORY_PROPERTY_HOST_COHERENT_BIT</code> is used, available memory in host domain is automatically made visible to host domain, and any host write is automatically made available to host domain.

If `srcQueueFamilyIndex` is not equal to `dstQueueFamilyIndex`, and `srcQueueFamilyIndex` is equal to the current queue family, then the memory barrier defines a [queue family release operation](#) for the specified buffer range, and the second synchronization scope of the calling command does not apply to this operation.

If `dstQueueFamilyIndex` is not equal to `srcQueueFamilyIndex`, and `dstQueueFamilyIndex` is equal to the current queue family, then the memory barrier defines a [queue family acquire operation](#) for the specified buffer range, and the first synchronization scope of the calling command does not apply to this operation.

Valid Usage

- VUID-VkBufferMemoryBarrier-offset-01187
`offset` **must** be less than the size of `buffer`
- VUID-VkBufferMemoryBarrier-size-01188
If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be greater than 0
- VUID-VkBufferMemoryBarrier-size-01189
If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be less than or equal to than the size of `buffer` minus `offset`
- VUID-VkBufferMemoryBarrier-buffer-01931

If **buffer** is non-sparse then it **must** be bound completely and contiguously to a single **VkDeviceMemory** object

- VUID-VkBufferMemoryBarrier-buffer-09095

If **buffer** was created with a sharing mode of **VK_SHARING_MODE_EXCLUSIVE**, and **srcQueueFamilyIndex** and **dstQueueFamilyIndex** are not equal, **srcQueueFamilyIndex** **must** be **VK_QUEUE_FAMILY_EXTERNAL**, **VK_QUEUE_FAMILY_FOREIGN_EXT**, or a valid queue family

- VUID-VkBufferMemoryBarrier-buffer-09096

If **buffer** was created with a sharing mode of **VK_SHARING_MODE_EXCLUSIVE**, and **srcQueueFamilyIndex** and **dstQueueFamilyIndex** are not equal, **dstQueueFamilyIndex** **must** be **VK_QUEUE_FAMILY_EXTERNAL**, **VK_QUEUE_FAMILY_FOREIGN_EXT**, or a valid queue family

- VUID-VkBufferMemoryBarrier-srcQueueFamilyIndex-09099

If the **VK_EXT_queue_family_foreign** extension is not enabled **srcQueueFamilyIndex** **must** not be **VK_QUEUE_FAMILY_FOREIGN_EXT**

- VUID-VkBufferMemoryBarrier-dstQueueFamilyIndex-09100

If the **VK_EXT_queue_family_foreign** extension is not enabled **dstQueueFamilyIndex** **must** not be **VK_QUEUE_FAMILY_FOREIGN_EXT**

- VUID-VkBufferMemoryBarrier-None-09049

If the **synchronization2** feature is not enabled, and **buffer** was created with a sharing mode of **VK_SHARING_MODE_CONCURRENT**, at least one of **srcQueueFamilyIndex** and **dstQueueFamilyIndex** **must** be **VK_QUEUE_FAMILY_IGNORED**

- VUID-VkBufferMemoryBarrier-None-09050

If the **synchronization2** feature is not enabled, and **buffer** was created with a sharing mode of **VK_SHARING_MODE_CONCURRENT**, **srcQueueFamilyIndex** **must** be **VK_QUEUE_FAMILY_IGNORED** or **VK_QUEUE_FAMILY_EXTERNAL**

- VUID-VkBufferMemoryBarrier-None-09051

If the **synchronization2** feature is not enabled, and **buffer** was created with a sharing mode of **VK_SHARING_MODE_CONCURRENT**, **dstQueueFamilyIndex** **must** be **VK_QUEUE_FAMILY_IGNORED** or **VK_QUEUE_FAMILY_EXTERNAL**

Valid Usage (Implicit)

- VUID-VkBufferMemoryBarrier-sType-sType

sType **must** be **VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER**

- VUID-VkBufferMemoryBarrier-pNext-pNext

pNext **must** be **NULL**

- VUID-VkBufferMemoryBarrier-buffer-parameter

buffer **must** be a valid **VkBuffer** handle

VK_WHOLE_SIZE is a special value indicating that the entire remaining length of a buffer or allocation following a given **offset** should be used. It **can** be specified for **VkBufferMemoryBarrier::size**, **vkMapMemory::size**, and other similar structures.

```
#define VK_WHOLE_SIZE
```

```
(~0ULL)
```

7.7.3. Image Memory Barriers

Image memory barriers only apply to memory accesses involving a specific image subresource range. That is, a memory dependency formed from an image memory barrier is [scoped](#) to access via the specified image subresource range. Image memory barriers **can** also be used to define [image layout transitions](#) or a [queue family ownership transfer](#) for the specified image subresource range.

The `VkImageMemoryBarrier2` structure is defined as:

```
typedef struct VkImageMemoryBarrier2 {
    VkStructureType      sType;
    const void*          pNext;
    VkPipelineStageFlags2 srcStageMask;
    VkAccessFlags2        srcAccessMask;
    VkPipelineStageFlags2 dstStageMask;
    VkAccessFlags2        dstAccessMask;
    VkImageLayout         oldLayout;
    VkImageLayout         newLayout;
    uint32_t              srcQueueFamilyIndex;
    uint32_t              dstQueueFamilyIndex;
    VkImage               image;
    VkImageSubresourceRange subresourceRange;
} VkImageMemoryBarrier2;
```

```
// Provided by VK_KHR_synchronization2
typedef VkImageMemoryBarrier2 VkImageMemoryBarrier2KHR;
```

- `sType` is a [VkStructureType](#) value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcStageMask` is a [VkPipelineStageFlags2](#) mask of pipeline stages to be included in the [first synchronization scope](#).
- `srcAccessMask` is a [VkAccessFlags2](#) mask of access flags to be included in the [first access scope](#).
- `dstStageMask` is a [VkPipelineStageFlags2](#) mask of pipeline stages to be included in the [second synchronization scope](#).
- `dstAccessMask` is a [VkAccessFlags2](#) mask of access flags to be included in the [second access scope](#).
- `oldLayout` is the old layout in an [image layout transition](#).
- `newLayout` is the new layout in an [image layout transition](#).
- `srcQueueFamilyIndex` is the source queue family for a [queue family ownership transfer](#).
- `dstQueueFamilyIndex` is the destination queue family for a [queue family ownership transfer](#).

- **image** is a handle to the image affected by this barrier.
- **subresourceRange** describes the **image subresource range** within **image** that is affected by this barrier.

This structure defines a **memory dependency** limited to an image subresource range, and **can** define a **queue family ownership transfer operation** and **image layout transition** for that subresource range.

The first **synchronization scope** and **access scope** described by this structure include only operations and memory accesses specified by the source stage mask and the source access mask.

The second **synchronization scope** and **access scope** described by this structure include only operations and memory accesses specified by the destination stage mask and the destination access mask.

Both **access scopes** are limited to only memory accesses to **image** in the subresource range defined by **subresourceRange**.

If **image** was created with **VK_SHARING_MODE_EXCLUSIVE**, and **srcQueueFamilyIndex** is not equal to **dstQueueFamilyIndex**, this memory barrier defines a **queue family ownership transfer operation**. When executed on a queue in the family identified by **srcQueueFamilyIndex**, this barrier defines a **queue family release operation** for the specified image subresource range, and the second synchronization scope does not apply to this operation. When executed on a queue in the family identified by **dstQueueFamilyIndex**, this barrier defines a **queue family acquire operation** for the specified image subresource range, and the first synchronization scope does not apply to this operation.

A **queue family ownership transfer operation** is also defined if the values are not equal, and either is one of the special queue family values reserved for external memory ownership transfers, as described in **Queue Family Ownership Transfer**. A **queue family release operation** is defined when **dstQueueFamilyIndex** is one of those values, and a **queue family acquire operation** is defined when **srcQueueFamilyIndex** is one of those values.

If **oldLayout** is not equal to **newLayout**, then the memory barrier defines an **image layout transition** for the specified image subresource range. If this memory barrier defines a **queue family ownership transfer operation**, the layout transition is only executed once between the queues.

NOTE

When the old and new layout are equal, the layout values are ignored - data is preserved no matter what values are specified, or what layout the image is currently in.

If **image** has a **multi-planar format** and the image is *disjoint*, then including **VK_IMAGE_ASPECT_COLOR_BIT** in the **aspectMask** member of **subresourceRange** is equivalent to including **VK_IMAGE_ASPECT_PLANE_0_BIT**, **VK_IMAGE_ASPECT_PLANE_1_BIT**, and (for three-plane formats only) **VK_IMAGE_ASPECT_PLANE_2_BIT**.

Valid Usage

- VUID-VkImageMemoryBarrier2-srcStageMask-03929

If the `geometryShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`

- VUID-VkImageMemoryBarrier2-srcStageMask-03930

If the `tessellationShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`

- VUID-VkImageMemoryBarrier2-srcStageMask-07317

If the `attachmentFragmentShadingRate` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`

- VUID-VkImageMemoryBarrier2-srcAccessMask-03900

If `srcAccessMask` includes `VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkImageMemoryBarrier2-srcAccessMask-03901

If `srcAccessMask` includes `VK_ACCESS_2_INDEX_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkImageMemoryBarrier2-srcAccessMask-03902

If `srcAccessMask` includes `VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT`, `VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkImageMemoryBarrier2-srcAccessMask-03903

If `srcAccessMask` includes `VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT`, `VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkImageMemoryBarrier2-srcAccessMask-03904

If `srcAccessMask` includes `VK_ACCESS_2_UNIFORM_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkImageMemoryBarrier2-srcAccessMask-03905

If `srcAccessMask` includes `VK_ACCESS_2_SHADER_SAMPLED_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkImageMemoryBarrier2-srcAccessMask-03906

If `srcAccessMask` includes `VK_ACCESS_2_SHADER_STORAGE_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkImageMemoryBarrier2-srcAccessMask-03907

If `srcAccessMask` includes `VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkImageMemoryBarrier2-srcAccessMask-07454
If `srcAccessMask` includes `VK_ACCESS_2_SHADER_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages
- VUID-VkImageMemoryBarrier2-srcAccessMask-03909
If `srcAccessMask` includes `VK_ACCESS_2_SHADER_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages
- VUID-VkImageMemoryBarrier2-srcAccessMask-03910
If `srcAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkImageMemoryBarrier2-srcAccessMask-03911
If `srcAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkImageMemoryBarrier2-srcAccessMask-03912
If `srcAccessMask` includes `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkImageMemoryBarrier2-srcAccessMask-03913
If `srcAccessMask` includes `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkImageMemoryBarrier2-srcAccessMask-03914
If `srcAccessMask` includes `VK_ACCESS_2_TRANSFER_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COPY_BIT`, `VK_PIPELINE_STAGE_2_BLIT_BIT`, `VK_PIPELINE_STAGE_2_RESOLVE_BIT`, `VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkImageMemoryBarrier2-srcAccessMask-03915
If `srcAccessMask` includes `VK_ACCESS_2_TRANSFER_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COPY_BIT`, `VK_PIPELINE_STAGE_2_BLIT_BIT`, `VK_PIPELINE_STAGE_2_RESOLVE_BIT`, `VK_PIPELINE_STAGE_2_CLEAR_BIT`, `VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkImageMemoryBarrier2-srcAccessMask-03916
If `srcAccessMask` includes `VK_ACCESS_2_HOST_READ_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_HOST_BIT`
- VUID-VkImageMemoryBarrier2-srcAccessMask-03917
If `srcAccessMask` includes `VK_ACCESS_2_HOST_WRITE_BIT`, `srcStageMask` **must** include `VK_PIPELINE_STAGE_2_HOST_BIT`
- VUID-VkImageMemoryBarrier2-srcAccessMask-03926
If `srcAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT`,

`srcStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkImageMemoryBarrier2-dstStageMask-03929
If the `geometryShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`
- VUID-VkImageMemoryBarrier2-dstStageMask-03930
If the `tessellationShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-VkImageMemoryBarrier2-dstStageMask-07317
If the `attachmentFragmentShadingRate` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-VkImageMemoryBarrier2-dstAccessMask-03900
If `dstAccessMask` includes `VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkImageMemoryBarrier2-dstAccessMask-03901
If `dstAccessMask` includes `VK_ACCESS_2_INDEX_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkImageMemoryBarrier2-dstAccessMask-03902
If `dstAccessMask` includes `VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT`, `VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkImageMemoryBarrier2-dstAccessMask-03903
If `dstAccessMask` includes `VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT`, `VK_PIPELINE_STAGE_2_SUBPASS_SHADER_BIT_HUAWEI`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`
- VUID-VkImageMemoryBarrier2-dstAccessMask-03904
If `dstAccessMask` includes `VK_ACCESS_2_UNIFORM_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages
- VUID-VkImageMemoryBarrier2-dstAccessMask-03905
If `dstAccessMask` includes `VK_ACCESS_2_SHADER_SAMPLED_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages
- VUID-VkImageMemoryBarrier2-dstAccessMask-03906
If `dstAccessMask` includes `VK_ACCESS_2_SHADER_STORAGE_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages
- VUID-VkImageMemoryBarrier2-dstAccessMask-03907

If `dstAccessMask` includes `VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkImageMemoryBarrier2-dstAccessMask-07454

If `dstAccessMask` includes `VK_ACCESS_2_SHADER_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkImageMemoryBarrier2-dstAccessMask-03909

If `dstAccessMask` includes `VK_ACCESS_2_SHADER_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`, or one of the `VK_PIPELINE_STAGE_*_SHADER_BIT` stages

- VUID-VkImageMemoryBarrier2-dstAccessMask-03910

If `dstAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkImageMemoryBarrier2-dstAccessMask-03911

If `dstAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkImageMemoryBarrier2-dstAccessMask-03912

If `dstAccessMask` includes `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkImageMemoryBarrier2-dstAccessMask-03913

If `dstAccessMask` includes `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkImageMemoryBarrier2-dstAccessMask-03914

If `dstAccessMask` includes `VK_ACCESS_2_TRANSFER_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_COPY_BIT`, `VK_PIPELINE_STAGE_2_BLIT_BIT`, `VK_PIPELINE_STAGE_2_RESOLVE_BIT`, `VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkImageMemoryBarrier2-dstAccessMask-03915

If `dstAccessMask` includes `VK_ACCESS_2_TRANSFER_WRITE_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_COPY_BIT`, `VK_PIPELINE_STAGE_2_BLIT_BIT`, `VK_PIPELINE_STAGE_2_RESOLVE_BIT`, `VK_PIPELINE_STAGE_2_CLEAR_BIT`, `VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkImageMemoryBarrier2-dstAccessMask-03916

If `dstAccessMask` includes `VK_ACCESS_2_HOST_READ_BIT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_HOST_BIT`

- VUID-VkImageMemoryBarrier2-dstAccessMask-03917

If `dstAccessMask` includes `VK_ACCESS_2_HOST_WRITE_BIT`, `dstStageMask` **must** include

VK_PIPELINE_STAGE_2_HOST_BIT

- VUID-VkImageMemoryBarrier2-dstAccessMask-03926

If `dstAccessMask` includes `VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT`, `dstStageMask` **must** include `VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT`, `VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT`, or `VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT`

- VUID-VkImageMemoryBarrier2-oldLayout-01208

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`

- VUID-VkImageMemoryBarrier2-oldLayout-01209

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-VkImageMemoryBarrier2-oldLayout-01210

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-VkImageMemoryBarrier2-oldLayout-01211

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_SAMPLED_BIT` or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`

- VUID-VkImageMemoryBarrier2-oldLayout-01212

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT`

- VUID-VkImageMemoryBarrier2-oldLayout-01213

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT`

- VUID-VkImageMemoryBarrier2-oldLayout-01197

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), `oldLayout` **must** be `VK_IMAGE_LAYOUT_UNDEFINED` or the current layout of the image subresources affected by the barrier

- VUID-VkImageMemoryBarrier2-newLayout-01198

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), `newLayout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`

- VUID-VkImageMemoryBarrier2-oldLayout-01658
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkImageMemoryBarrier2-oldLayout-01659
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-04065
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL` then `image` **must** have been created with at least one of `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_SAMPLED_BIT`, or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-04066
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT` set
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-04067
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL` then `image` **must** have been created with at least one of `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_SAMPLED_BIT`, or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-04068
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT` set
- VUID-VkImageMemoryBarrier2-synchronization2-07793
If the `synchronization2` feature is not enabled, `oldLayout` **must** not be `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR` or `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`
- VUID-VkImageMemoryBarrier2-synchronization2-07794
If the `synchronization2` feature is not enabled, `newLayout` **must** not be `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR` or `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-03938
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL`, `image` **must** have been created with `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` or `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-03939

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL`, `image` **must** have been created with at least one of `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_SAMPLED_BIT`, or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`

- VUID-VkImageMemoryBarrier2-oldLayout-02088

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR` then `image` **must** have been created with `VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR` set

- VUID-VkImageMemoryBarrier2-image-09117

If `image` was created with a sharing mode of `VK_SHARING_MODE_EXCLUSIVE`, and `srcQueueFamilyIndex` and `dstQueueFamilyIndex` are not equal, `srcQueueFamilyIndex` **must** be `VK_QUEUE_FAMILY_EXTERNAL`, `VK_QUEUE_FAMILY_FOREIGN_EXT`, or a valid queue family

- VUID-VkImageMemoryBarrier2-image-09118

If `image` was created with a sharing mode of `VK_SHARING_MODE_EXCLUSIVE`, and `srcQueueFamilyIndex` and `dstQueueFamilyIndex` are not equal, `dstQueueFamilyIndex` **must** be `VK_QUEUE_FAMILY_EXTERNAL`, `VK_QUEUE_FAMILY_FOREIGN_EXT`, or a valid queue family

- VUID-VkImageMemoryBarrier2-srcQueueFamilyIndex-09121

If the [VK_EXT_queue_family_foreign](#) extension is not enabled `srcQueueFamilyIndex` **must** not be `VK_QUEUE_FAMILY_FOREIGN_EXT`

- VUID-VkImageMemoryBarrier2-dstQueueFamilyIndex-09122

If the [VK_EXT_queue_family_foreign](#) extension is not enabled `dstQueueFamilyIndex` **must** not be `VK_QUEUE_FAMILY_FOREIGN_EXT`

- VUID-VkImageMemoryBarrier2-subresourceRange-01486

`subresourceRange.baseMipLevel` **must** be less than the `mipLevels` specified in [VkImageCreateInfo](#) when `image` was created

- VUID-VkImageMemoryBarrier2-subresourceRange-01724

If `subresourceRange.levelCount` is not `VK_REMAINING_MIP_LEVELS`, `subresourceRange.baseMipLevel + subresourceRange.levelCount` **must** be less than or equal to the `mipLevels` specified in [VkImageCreateInfo](#) when `image` was created

- VUID-VkImageMemoryBarrier2-subresourceRange-01488

`subresourceRange.baseArrayLayer` **must** be less than the `arrayLayers` specified in [VkImageCreateInfo](#) when `image` was created

- VUID-VkImageMemoryBarrier2-subresourceRange-01725

If `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `subresourceRange.baseArrayLayer + subresourceRange.layerCount` **must** be less than or equal to the `arrayLayers` specified in [VkImageCreateInfo](#) when `image` was created

- VUID-VkImageMemoryBarrier2-image-01932

If `image` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-VkImageMemoryBarrier2-image-09241

If `image` has a color format that is single-plane, then the `aspectMask` member of

`subresourceRange` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`

- VUID-VkImageMemoryBarrier2-image-09242

If `image` has a color format and is not *disjoint*, then the `aspectMask` member of `subresourceRange` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`

- VUID-VkImageMemoryBarrier2-image-01672

If `image` has a `multi-planar format` and the image is *disjoint*, then the `aspectMask` member of `subresourceRange` **must** include at least one `multi-planar aspect mask` bit or `VK_IMAGE_ASPECT_COLOR_BIT`

- VUID-VkImageMemoryBarrier2-image-03320

If `image` has a depth/stencil format with both depth and stencil and the `separateDepthStencilLayouts` feature is not enabled, then the `aspectMask` member of `subresourceRange` **must** include both `VK_IMAGE_ASPECT_DEPTH_BIT` and `VK_IMAGE_ASPECT_STENCIL_BIT`

- VUID-VkImageMemoryBarrier2-image-03319

If `image` has a depth/stencil format with both depth and stencil and the `separateDepthStencilLayouts` feature is enabled, then the `aspectMask` member of `subresourceRange` **must** include either or both `VK_IMAGE_ASPECT_DEPTH_BIT` and `VK_IMAGE_ASPECT_STENCIL_BIT`

- VUID-VkImageMemoryBarrier2-image-10749

If `image` has a depth-only format then the `aspectMask` member of `subresourceRange` **must** be `VK_IMAGE_ASPECT_DEPTH_BIT`

- VUID-VkImageMemoryBarrier2-image-10750

If `image` has a stencil-only format then the `aspectMask` member of `subresourceRange` **must** be `VK_IMAGE_ASPECT_STENCIL_BIT`

- VUID-VkImageMemoryBarrier2-aspectMask-08702

If the `aspectMask` member of `subresourceRange` includes `VK_IMAGE_ASPECT_DEPTH_BIT`, `oldLayout` and `newLayout` **must** not be one of `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`

- VUID-VkImageMemoryBarrier2-aspectMask-08703

If the `aspectMask` member of `subresourceRange` includes `VK_IMAGE_ASPECT_STENCIL_BIT`, `oldLayout` and `newLayout` **must** not be one of `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`

- VUID-VkImageMemoryBarrier2-subresourceRange-09601

`subresourceRange.aspectMask` **must** be valid for the `format` the `image` was created with

- VUID-VkImageMemoryBarrier2-srcStageMask-03854

If either `srcStageMask` or `dstStageMask` includes `VK_PIPELINE_STAGE_2_HOST_BIT`, `srcQueueFamilyIndex` and `dstQueueFamilyIndex` **must** be equal

- VUID-VkImageMemoryBarrier2-srcStageMask-03855

If `srcStageMask` includes `VK_PIPELINE_STAGE_2_HOST_BIT`, and `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a `queue family ownership transfer` or `oldLayout` and `newLayout` define an `image layout transition`, `oldLayout` **must** be one of `VK_IMAGE_LAYOUT_PREINITIALIZED`, `VK_IMAGE_LAYOUT_UNDEFINED`, or `VK_IMAGE_LAYOUT_GENERAL`

Valid Usage (Implicit)

- VUID-VkImageMemoryBarrier2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER_2`
- VUID-VkImageMemoryBarrier2-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkSampleLocationsInfoEXT`
- VUID-VkImageMemoryBarrier2-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkImageMemoryBarrier2-srcStageMask-parameter
`srcStageMask` **must** be a valid combination of `VkPipelineStageFlagBits2` values
- VUID-VkImageMemoryBarrier2-srcAccessMask-parameter
`srcAccessMask` **must** be a valid combination of `VkAccessFlagBits2` values
- VUID-VkImageMemoryBarrier2-dstStageMask-parameter
`dstStageMask` **must** be a valid combination of `VkPipelineStageFlagBits2` values
- VUID-VkImageMemoryBarrier2-dstAccessMask-parameter
`dstAccessMask` **must** be a valid combination of `VkAccessFlagBits2` values
- VUID-VkImageMemoryBarrier2-oldLayout-parameter
`oldLayout` **must** be a valid `VkImageLayout` value
- VUID-VkImageMemoryBarrier2-newLayout-parameter
`newLayout` **must** be a valid `VkImageLayout` value
- VUID-VkImageMemoryBarrier2-image-parameter
`image` **must** be a valid `VkImage` handle
- VUID-VkImageMemoryBarrier2-subresourceRange-parameter
`subresourceRange` **must** be a valid `VkImageSubresourceRange` structure

The `VkImageMemoryBarrier` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageMemoryBarrier {
    VkStructureType    sType;
    const void*        pNext;
    VkAccessFlags       srcAccessMask;
    VkAccessFlags       dstAccessMask;
    VkImageLayout       oldLayout;
    VkImageLayout       newLayout;
    uint32_t            srcQueueFamilyIndex;
    uint32_t            dstQueueFamilyIndex;
    VkImage             image;
    VkImageSubresourceRange subresourceRange;
} VkImageMemoryBarrier;
```

- `sType` is a `VkStructureType` value identifying this structure.

- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcAccessMask` is a bitmask of `VkAccessFlagBits` specifying a [source access mask](#).
- `dstAccessMask` is a bitmask of `VkAccessFlagBits` specifying a [destination access mask](#).
- `oldLayout` is the old layout in an [image layout transition](#).
- `newLayout` is the new layout in an [image layout transition](#).
- `srcQueueFamilyIndex` is the source queue family for a [queue family ownership transfer](#).
- `dstQueueFamilyIndex` is the destination queue family for a [queue family ownership transfer](#).
- `image` is a handle to the image affected by this barrier.
- `subresourceRange` describes the [image subresource range](#) within `image` that is affected by this barrier.

The first [access scope](#) is limited to access to memory through the specified image subresource range, via access types in the [source access mask](#) specified by `srcAccessMask`. If `srcAccessMask` includes `VK_ACCESS_HOST_WRITE_BIT`, memory writes performed by that access type are also made visible, as that access type is not performed through a resource.

The second [access scope](#) is limited to access to memory through the specified image subresource range, via access types in the [destination access mask](#) specified by `dstAccessMask`. If `dstAccessMask` includes `VK_ACCESS_HOST_WRITE_BIT` or `VK_ACCESS_HOST_READ_BIT`, available memory writes are also made visible to accesses of those types, as those access types are not performed through a resource.

If `srcQueueFamilyIndex` is not equal to `dstQueueFamilyIndex`, and `srcQueueFamilyIndex` is equal to the current queue family, then the memory barrier defines a [queue family release operation](#) for the specified image subresource range, and the second synchronization scope of the calling command does not apply to this operation.

If `dstQueueFamilyIndex` is not equal to `srcQueueFamilyIndex`, and `dstQueueFamilyIndex` is equal to the current queue family, then the memory barrier defines a [queue family acquire operation](#) for the specified image subresource range, and the first synchronization scope of the calling command does not apply to this operation.

If the `synchronization2` feature is not enabled or `oldLayout` is not equal to `newLayout`, `oldLayout` and `newLayout` define an [image layout transition](#) for the specified image subresource range.

NOTE

If the `synchronization2` feature is enabled, when the old and new layout are equal, the layout values are ignored - data is preserved no matter what values are specified, or what layout the image is currently in.

If `image` has a [multi-planar format](#) and the image is *disjoint*, then including `VK_IMAGE_ASPECT_COLOR_BIT` in the `aspectMask` member of `subresourceRange` is equivalent to including `VK_IMAGE_ASPECT_PLANE_0_BIT`, `VK_IMAGE_ASPECT_PLANE_1_BIT`, and (for three-plane formats only) `VK_IMAGE_ASPECT_PLANE_2_BIT`.

Valid Usage

- VUID-VkImageMemoryBarrier-oldLayout-01208
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
- VUID-VkImageMemoryBarrier-oldLayout-01209
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkImageMemoryBarrier-oldLayout-01210
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkImageMemoryBarrier-oldLayout-01211
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_SAMPLED_BIT` or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`
- VUID-VkImageMemoryBarrier-oldLayout-01212
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT`
- VUID-VkImageMemoryBarrier-oldLayout-01213
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT`
- VUID-VkImageMemoryBarrier-oldLayout-01197
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), `oldLayout` **must** be `VK_IMAGE_LAYOUT_UNDEFINED` or the current layout of the image subresources affected by the barrier
- VUID-VkImageMemoryBarrier-newLayout-01198
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), `newLayout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`
- VUID-VkImageMemoryBarrier-oldLayout-01658
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is

VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL then **image must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT

- VUID-VkImageMemoryBarrier-oldLayout-01659

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL then **image must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT

- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-04065

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL then **image must** have been created with at least one of VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, VK_IMAGE_USAGE_SAMPLED_BIT, or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT

- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-04066

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL then **image must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT set

- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-04067

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL then **image must** have been created with at least one of VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, VK_IMAGE_USAGE_SAMPLED_BIT, or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT

- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-04068

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL then **image must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT set

- VUID-VkImageMemoryBarrier-synchronization2-07793

If the `synchronization2` feature is not enabled, `oldLayout` **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR

- VUID-VkImageMemoryBarrier-synchronization2-07794

If the `synchronization2` feature is not enabled, `newLayout` **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR

- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-03938

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL, **image must** have been created with VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT or VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT

- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-03939

If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL, **image must** have been created with at least one of

VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, VK_IMAGE_USAGE_SAMPLED_BIT, or
VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT

- VUID-VkImageMemoryBarrier-oldLayout-02088
If `srcQueueFamilyIndex` and `dstQueueFamilyIndex` define a [queue family ownership transfer](#) or `oldLayout` and `newLayout` define an [image layout transition](#), and `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR` then `image` **must** have been created with `VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR` set
- VUID-VkImageMemoryBarrier-image-09117
If `image` was created with a sharing mode of `VK_SHARING_MODE_EXCLUSIVE`, and `srcQueueFamilyIndex` and `dstQueueFamilyIndex` are not equal, `srcQueueFamilyIndex` **must** be `VK_QUEUE_FAMILY_EXTERNAL`, `VK_QUEUE_FAMILY_FOREIGN_EXT`, or a valid queue family
- VUID-VkImageMemoryBarrier-image-09118
If `image` was created with a sharing mode of `VK_SHARING_MODE_EXCLUSIVE`, and `srcQueueFamilyIndex` and `dstQueueFamilyIndex` are not equal, `dstQueueFamilyIndex` **must** be `VK_QUEUE_FAMILY_EXTERNAL`, `VK_QUEUE_FAMILY_FOREIGN_EXT`, or a valid queue family
- VUID-VkImageMemoryBarrier-srcQueueFamilyIndex-09121
If the [VK_EXT_queue_family_foreign](#) extension is not enabled `srcQueueFamilyIndex` **must** not be `VK_QUEUE_FAMILY_FOREIGN_EXT`
- VUID-VkImageMemoryBarrier-dstQueueFamilyIndex-09122
If the [VK_EXT_queue_family_foreign](#) extension is not enabled `dstQueueFamilyIndex` **must** not be `VK_QUEUE_FAMILY_FOREIGN_EXT`
- VUID-VkImageMemoryBarrier-subresourceRange-01486
`subresourceRange.baseMipLevel` **must** be less than the `mipLevels` specified in [VkImageCreateInfo](#) when `image` was created
- VUID-VkImageMemoryBarrier-subresourceRange-01724
If `subresourceRange.levelCount` is not `VK_REMAINING_MIP_LEVELS`, `subresourceRange.baseMipLevel + subresourceRange.levelCount` **must** be less than or equal to the `mipLevels` specified in [VkImageCreateInfo](#) when `image` was created
- VUID-VkImageMemoryBarrier-subresourceRange-01488
`subresourceRange.baseArrayLayer` **must** be less than the `arrayLayers` specified in [VkImageCreateInfo](#) when `image` was created
- VUID-VkImageMemoryBarrier-subresourceRange-01725
If `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `subresourceRange.baseArrayLayer + subresourceRange.layerCount` **must** be less than or equal to the `arrayLayers` specified in [VkImageCreateInfo](#) when `image` was created
- VUID-VkImageMemoryBarrier-image-01932
If `image` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkImageMemoryBarrier-image-09241
If `image` has a color format that is single-plane, then the `aspectMask` member of `subresourceRange` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkImageMemoryBarrier-image-09242
If `image` has a color format and is not *disjoint*, then the `aspectMask` member of

subresourceRange **must** be VK_IMAGE_ASPECT_COLOR_BIT

- VUID-VkImageMemoryBarrier-image-01672

If **image** has a multi-planar format and the image is *disjoint*, then the aspectMask member of subresourceRange **must** include at least one multi-planar aspect mask bit or VK_IMAGE_ASPECT_COLOR_BIT

- VUID-VkImageMemoryBarrier-image-03320

If **image** has a depth/stencil format with both depth and stencil and the separateDepthStencilLayouts feature is not enabled, then the aspectMask member of subresourceRange **must** include both VK_IMAGE_ASPECT_DEPTH_BIT and VK_IMAGE_ASPECT_STENCIL_BIT

- VUID-VkImageMemoryBarrier-image-03319

If **image** has a depth/stencil format with both depth and stencil and the separateDepthStencilLayouts feature is enabled, then the aspectMask member of subresourceRange **must** include either or both VK_IMAGE_ASPECT_DEPTH_BIT and VK_IMAGE_ASPECT_STENCIL_BIT

- VUID-VkImageMemoryBarrier-image-10749

If **image** has a depth-only format then the aspectMask member of subresourceRange **must** be VK_IMAGE_ASPECT_DEPTH_BIT

- VUID-VkImageMemoryBarrier-image-10750

If **image** has a stencil-only format then the aspectMask member of subresourceRange **must** be VK_IMAGE_ASPECT_STENCIL_BIT

- VUID-VkImageMemoryBarrier-aspectMask-08702

If the aspectMask member of subresourceRange includes VK_IMAGE_ASPECT_DEPTH_BIT, oldLayout and newLayout **must** not be one of VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkImageMemoryBarrier-aspectMask-08703

If the aspectMask member of subresourceRange includes VK_IMAGE_ASPECT_STENCIL_BIT, oldLayout and newLayout **must** not be one of VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL

- VUID-VkImageMemoryBarrier-subresourceRange-09601

subresourceRange.aspectMask **must** be valid for the format the **image** was created with

- VUID-VkImageMemoryBarrier-None-09052

If the synchronization2 feature is not enabled, and **image** was created with a sharing mode of VK_SHARING_MODE_CONCURRENT, at least one of srcQueueFamilyIndex and dstQueueFamilyIndex **must** be VK_QUEUE_FAMILY_IGNORED

- VUID-VkImageMemoryBarrier-None-09053

If the synchronization2 feature is not enabled, and **image** was created with a sharing mode of VK_SHARING_MODE_CONCURRENT, srcQueueFamilyIndex **must** be VK_QUEUE_FAMILY_IGNORED or VK_QUEUE_FAMILY_EXTERNAL

- VUID-VkImageMemoryBarrier-None-09054

If the synchronization2 feature is not enabled, and **image** was created with a sharing mode of VK_SHARING_MODE_CONCURRENT, dstQueueFamilyIndex **must** be VK_QUEUE_FAMILY_IGNORED or VK_QUEUE_FAMILY_EXTERNAL

Valid Usage (Implicit)

- VUID-VkImageMemoryBarrier-sType-sType
sType must be `VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER`
- VUID-VkImageMemoryBarrier-pNext-pNext
pNext must be `NULL` or a pointer to a valid instance of `VkSampleLocationsInfoEXT`
- VUID-VkImageMemoryBarrier-sType-unique
The **sType** value of each structure in the **pNext** chain **must** be unique
- VUID-VkImageMemoryBarrier-oldLayout-parameter
oldLayout must be a valid `VkImageLayout` value
- VUID-VkImageMemoryBarrier-newLayout-parameter
newLayout must be a valid `VkImageLayout` value
- VUID-VkImageMemoryBarrier-image-parameter
image must be a valid `VkImage` handle
- VUID-VkImageMemoryBarrier-subresourceRange-parameter
subresourceRange must be a valid `VkImageSubresourceRange` structure

7.7.4. Queue Family Ownership Transfer

Resources created with a `VkSharingMode` of `VK_SHARING_MODE_EXCLUSIVE` **must** have their ownership explicitly transferred from one queue family to another in order to access their content in a well-defined manner on a queue in a different queue family.

The special queue family index `VK_QUEUE_FAMILY_IGNORED` indicates that a queue family parameter or member is ignored.

```
#define VK_QUEUE_FAMILY_IGNORED          (~0U)
```

Resources shared with external APIs or instances using external memory **must** also explicitly manage ownership transfers between local and external queues (or equivalent constructs in external APIs) regardless of the `VkSharingMode` specified when creating them.

The special queue family index `VK_QUEUE_FAMILY_EXTERNAL` represents any queue external to the resource's current Vulkan instance, as long as the queue uses the same underlying device group or physical device, and the same driver version as the resource's `VkDevice`, as indicated by `VkPhysicalDeviceIDProperties::deviceUUID` and `VkPhysicalDeviceIDProperties::driverUUID`.

```
#define VK_QUEUE_FAMILY_EXTERNAL          (~1U)
```

The special queue family index `VK_QUEUE_FAMILY_FOREIGN_EXT` represents any queue external to the resource's current Vulkan instance, regardless of the queue's underlying physical device or driver version. This includes, for example, queues for fixed-function image processing devices, media codec devices, and display devices, as well as all queues that use the same underlying device group

or physical device, and the same driver version as the resource's [VkDevice](#).

```
#define VK_QUEUE_FAMILY_FOREIGN_EXT    (~2U)
```

If memory dependencies are correctly expressed between uses of such a resource between two queues in different families, but no ownership transfer is defined, the contents of that resource are undefined for any read accesses performed by the second queue family.

NOTE	If an application does not need the contents of a resource to remain valid when transferring from one queue family to another, then the ownership transfer should be skipped.
NOTE	Applications should expect transfers to/from <code>VK_QUEUE_FAMILY_FOREIGN_EXT</code> to be more expensive than transfers to/from <code>VK_QUEUE_FAMILY_EXTERNAL_KHR</code> .

A queue family ownership transfer consists of two distinct parts:

1. Release exclusive ownership from the source queue family
2. Acquire exclusive ownership for the destination queue family

An application **must** ensure that these operations occur in the correct order by defining an execution dependency between them, e.g. using a semaphore.

A *release operation* is used to release exclusive ownership of a range of a buffer or image subresource range. A release operation is defined by executing a [buffer memory barrier](#) (for a buffer range) or an [image memory barrier](#) (for an image subresource range) using a pipeline barrier command, on a queue from the source queue family. The `srcQueueFamilyIndex` parameter of the barrier **must** be the source queue family index, and the `dstQueueFamilyIndex` parameter to the destination queue family index. The destination access mask is ignored for such a barrier, such that no visibility operation is executed - the value of this mask does not affect the validity of the barrier. The release operation happens-after the availability operation. `dstStageMask` is also ignored for such a barrier as defined by [buffer memory ownership transfer](#) and [image memory ownership transfer](#). After a release operation is performed, the contents and image layout (if applicable) of the released resource are undefined until a matching acquire operation is performed.

NOTE	It is valid to never call the acquire operation after a release, and instead simply start using the resource on any queue (even the releasing queue), but the contents should be reinitialized before being read. In the case of images, an image layout transition away from <code>VK_IMAGE_LAYOUT_UNDEFINED</code> is also required. Executing an acquire operation after this without another release is invalid.

An *acquire operation* is used to acquire exclusive ownership of a range of a buffer or image subresource range. An acquire operation is defined by executing a [buffer memory barrier](#) (for a buffer range) or an [image memory barrier](#) (for an image subresource range) using a pipeline barrier command, on a queue from the destination queue family. The buffer range or image subresource range specified in an acquire operation **must** match exactly that of a previous release operation. The `srcQueueFamilyIndex` parameter of the barrier **must** be the source queue family

index, and the `dstQueueFamilyIndex` parameter to the destination queue family index. The source access mask is ignored for such a barrier, such that no availability operation is executed - the value of this mask does not affect the validity of the barrier. The acquire operation happens-before the visibility operation. `srcStageMask` is also ignored for such a barrier as defined by [buffer memory ownership transfer](#) and [image memory ownership transfer](#). Applications **must** not execute an acquire operation on any resource where it has been modified in any way other than a matching release operation.

NOTE

Whilst it is not invalid to provide destination or source access masks for memory barriers used for release or acquire operations, respectively, they have no practical effect. Access after a release operation has undefined results, and so visibility for those accesses has no practical effect. Similarly, write access before an acquire operation will produce undefined results for future access, so availability of those writes has no practical use. In an earlier version of the specification, these were required to match on both sides - but this was subsequently relaxed. These masks **should** be set to 0.

NOTE

To ensure that an acquire and release operation are valid, the release operation must happen-before the acquire operation. Often, semaphores are used for this directly, with the semaphore signaling after a release and then waiting before an acquire. `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT` is the only valid stage to wait on or wait for these operations, as the acquire and release operations do not occur in a defined stage.

If the transfer is via an image memory barrier, and an [image layout transition](#) is desired, then the values of `oldLayout` and `newLayout` in the *release operation's* memory barrier **must** be equal to values of `oldLayout` and `newLayout` in the *acquire operation's* memory barrier. Although the image layout transition is submitted twice, it will only be executed once. A layout transition specified in this way happens-after the *release operation* and happens-before the *acquire operation*.

If the values of `srcQueueFamilyIndex` and `dstQueueFamilyIndex` are equal, no ownership transfer is performed, and the barrier operates as if they were both set to `VK_QUEUE_FAMILY_IGNORED`.

Queue family ownership transfers **may** perform read and write accesses on all memory bound to the image subresource or buffer range, so applications **must** ensure that all memory writes have been made [available](#) before a queue family ownership transfer is executed. Available memory is automatically made visible to queue family release and acquire operations, and writes performed by those operations are automatically made available.

Once a queue family has acquired ownership of a buffer range or image subresource range of a `VK_SHARING_MODE_EXCLUSIVE` resource, its contents are undefined to other queue families unless ownership is transferred. The contents of any portion of another resource which aliases memory that is bound to the transferred buffer or image subresource range are undefined after a release or acquire operation.

NOTE

Because [events](#) **cannot** be used directly for inter-queue synchronization, and because `vkCmdSetEvent` does not have the queue family index or memory barrier parameters needed by a *release operation*, the release and acquire operations of a

queue family ownership transfer **can** only be performed using `vkCmdPipelineBarrier`.

7.8. Wait Idle Operations

To wait on the host for the completion of outstanding queue operations for a given queue, call:

```
// Provided by VK_VERSION_1_0
VkResult vkQueueWaitIdle(
    VkQueue queue);
```

- `queue` is the queue on which to wait.

`vkQueueWaitIdle` is equivalent to having submitted a valid fence to every previously executed `queue submission command` that accepts a fence, then waiting for all of those fences to signal using `vkWaitForFences` with an infinite timeout and `waitAll` set to `VK_TRUE`.

Even though `vkQueuePresentKHR` does not accept a fence, for the purpose of `vkQueueWaitIdle` it behaves as if it signals a fence payload that is managed by the implementation and has no associated `VkFence` handle. See the `vkQueuePresentKHR` documentation for details.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkQueueWaitIdle` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkQueueWaitIdle-queue-parameter `queue` **must** be a valid `VkQueue` handle

Host Synchronization

- Host access to `queue` **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
-	-	Any	-

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To wait on the host for the completion of outstanding queue operations for all queues on a given logical device, call:

```
// Provided by VK_VERSION_1_0
VkResult vkDeviceWaitIdle(
    VkDevice device);
```

- `device` is the logical device to idle.

`vkDeviceWaitIdle` is equivalent to calling `vkQueueWaitIdle` for all queues owned by `device`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkDeviceWaitIdle` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkDeviceWaitIdle-device-parameter `device` **must** be a valid `VkDevice` handle

Host Synchronization

- Host access to all `VkQueue` objects created from `device` **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_DEVICE_LOST`

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

7.9. Host Write Ordering Guarantees

When batches of command buffers are submitted to a queue via a [queue submission command](#), it defines a memory dependency with prior host operations, and execution of command buffers submitted to the queue.

The first [synchronization scope](#) includes execution of `vkQueueSubmit` on the host and anything that happened-before it, as defined by the host memory model.

NOTE

Some systems allow writes that do not directly integrate with the host memory model; these have to be synchronized by the application manually. One example of this is non-temporal store instructions on x86; to ensure these happen-before submission, applications should call `_mm_sfence()`.

The second [synchronization scope](#) includes all commands submitted in the same [queue submission](#), and all commands that occur later in [submission order](#).

The first [access scope](#) includes all host writes to mappable device memory that are available to the host memory domain.

The second [access scope](#) includes all memory access performed by the device.

7.10. Synchronization and Multiple Physical Devices

If a logical device includes more than one physical device, then fences, semaphores, and events all still have a single instance of the signaled state.

A fence becomes signaled when all physical devices complete the necessary queue operations.

Semaphore wait and signal operations all include a device index that is the sole physical device that performs the operation. These indices are provided in the [VkDeviceGroupSubmitInfo](#) structures. Semaphores are not exclusively owned by any physical device. For example, a semaphore can be signaled by one physical device and then waited on by a different physical device.

An event **can** only be waited on by the same physical device that signaled it (or the host).

7.11. Calibrated Timestamps

In order to be able to correlate the time a particular operation took place at on timelines of different time domains (e.g. a device operation vs. a host operation), Vulkan allows querying calibrated timestamps from multiple time domains.

To query calibrated timestamps from a set of time domains, call:

```
// Provided by VK_KHR_calibrated_timestamps
VkResult vkGetCalibratedTimestampsKHR(
    VkDevice          device,
    uint32_t          timestampCount,
    const VkCalibratedTimestampInfoKHR* pTimestampInfos,
    uint64_t*          pTimestamps,
    uint64_t*          pMaxDeviation);
```

- **device** is the logical device used to perform the query.
- **timestampCount** is the number of timestamps to query.
- **pTimestampInfos** is a pointer to an array of **timestampCount** [VkCalibratedTimestampInfoKHR](#) structures, describing the time domains the calibrated timestamps should be captured from.
- **pTimestamps** is a pointer to an array of **timestampCount** 64-bit unsigned integer values in which the requested calibrated timestamp values are returned.
- **pMaxDeviation** is a pointer to a 64-bit unsigned integer value in which the strictly positive maximum deviation, in nanoseconds, of the calibrated timestamp values is returned.

NOTE

The maximum deviation **may** vary between calls to [vkGetCalibratedTimestampsKHR](#) even for the same set of time domains due to implementation and platform specific reasons. It is the application's responsibility to assess whether the returned maximum deviation makes the timestamp values suitable for any particular purpose and **can** choose to re-issue the timestamp calibration call pursuing a lower deviation value.

Calibrated timestamp values **can** be extrapolated to estimate future coinciding timestamp values, however, depending on the nature of the time domains and other properties of the platform extrapolating values over a sufficiently long period of time **may** no longer be accurate enough to fit any particular purpose, so applications are expected to re-calibrate the timestamps on a regular basis.

Valid Usage

- VUID-vkGetCalibratedTimestampsKHR-timeDomain-09246
The **timeDomain** value of each [VkCalibratedTimestampInfoKHR](#) in **pTimestampInfos** **must** be unique

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is **VK_TRUE**, [vkGetCalibratedTimestampsKHR](#) **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage (Implicit)

- VUID-vkGetCalibratedTimestampsKHR-device-parameter

device **must** be a valid [VkDevice](#) handle

- VUID-vkGetCalibratedTimestampsKHR-pTimestampInfos-parameter
pTimestampInfos **must** be a valid pointer to an array of **timestampCount** valid [VkCalibratedTimestampInfoKHR](#) structures
- VUID-vkGetCalibratedTimestampsKHR-pTimestamps-parameter
pTimestamps **must** be a valid pointer to an array of **timestampCount** **uint64_t** values
- VUID-vkGetCalibratedTimestampsKHR-pMaxDeviation-parameter
pMaxDeviation **must** be a valid pointer to a **uint64_t** value
- VUID-vkGetCalibratedTimestampsKHR-timestampCount-arraylength
timestampCount **must** be greater than **0**

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The [VkCalibratedTimestampInfoKHR](#) structure is defined as:

```
// Provided by VK_KHR_calibrated_timestamps
typedef struct VkCalibratedTimestampInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkTimeDomainKHR    timeDomain;
} VkCalibratedTimestampInfoKHR;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **timeDomain** is a [VkTimeDomainKHR](#) value specifying the time domain from which the calibrated timestamp value should be returned.

Valid Usage

- VUID-VkCalibratedTimestampInfoKHR-timeDomain-02354
timeDomain **must** be one of the [VkTimeDomainKHR](#) values returned by [vkGetPhysicalDeviceCalibrateableTimeDomainsKHR](#)

Valid Usage (Implicit)

- VUID-VkCalibratedTimestampInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_CALIBRATED_TIMESTAMP_INFO_KHR`
- VUID-VkCalibratedTimestampInfoKHR-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkCalibratedTimestampInfoKHR-timeDomain-parameter
`timeDomain` **must** be a valid `VkTimeDomainKHR` value

The set of supported time domains consists of:

```
// Provided by VK_KHR_calibrated_timestamps
typedef enum VkTimeDomainKHR {
    VK_TIME_DOMAIN_DEVICE_KHR = 0,
    VK_TIME_DOMAIN_CLOCK_MONOTONIC_KHR = 1,
    VK_TIME_DOMAIN_CLOCK_MONOTONIC_RAW_KHR = 2,
    VK_TIME_DOMAIN_QUERY_PERFORMANCE_COUNTER_KHR = 3,
} VkTimeDomainKHR;
```

- `VK_TIME_DOMAIN_DEVICE_KHR` specifies the device time domain. Timestamp values in this time domain use the same units and are comparable with device timestamp values captured using `vkCmdWriteTimestamp` or `vkCmdWriteTimestamp2KHR` and are defined to be incrementing according to the `timestampPeriod` of the device.
- `VK_TIME_DOMAIN_CLOCK_MONOTONIC_KHR` specifies the `CLOCK_MONOTONIC` time domain available on POSIX platforms. Timestamp values in this time domain are in units of nanoseconds and are comparable with platform timestamp values captured using the POSIX `clock_gettime` API as computed by this example:

NOTE

An implementation supporting `VK_KHR_calibrated_timestamps` will use the same time domain for all its `VkQueue` so that timestamp values reported for `VK_TIME_DOMAIN_DEVICE_KHR` can be matched to any timestamp captured through `vkCmdWriteTimestamp` or `vkCmdWriteTimestamp2KHR`.

```
struct timespec tv;
clock_gettime(CLOCK_MONOTONIC, &tv);
return tv.tv_nsec + tv.tv_sec*1000000000ull;
```

- `VK_TIME_DOMAIN_CLOCK_MONOTONIC_RAW_KHR` specifies the `CLOCK_MONOTONIC_RAW` time domain available on POSIX platforms. Timestamp values in this time domain are in units of nanoseconds and are comparable with platform timestamp values captured using the POSIX `clock_gettime` API as computed by this example:

```
struct timespec tv;
clock_gettime(CLOCK_MONOTONIC_RAW, &tv);
```

```
return tv.tv_nsec + tv.tv_sec*1000000000ull;
```

- `VK_TIME_DOMAIN_QUERY_PERFORMANCE_COUNTER_KHR` specifies the performance counter (QPC) time domain available on Windows. Timestamp values in this time domain are in the same units as those provided by the Windows QueryPerformanceCounter API and are comparable with platform timestamp values captured using that API as computed by this example:

```
LARGE_INTEGER counter;  
QueryPerformanceCounter(&counter);  
return counter.QuadPart;
```

Chapter 8. Render Pass

Draw commands **must** be recorded within a *render pass instance*. Each render pass instance defines a set of image resources, referred to as *attachments*, used during rendering.

A render pass object represents a collection of attachments, subpasses, and dependencies between the subpasses, and describes how the attachments are used over the course of the subpasses.

Render passes are represented by **VkRenderPass** handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkRenderPass)
```

An *attachment description* describes the properties of an attachment including its format, sample count, and how its contents are treated at the beginning and end of each render pass instance.

A *subpass* represents a phase of rendering that reads and writes a subset of the attachments in a render pass. Rendering commands are recorded into a particular subpass of a render pass instance.

A *subpass description* describes the subset of attachments that is involved in the execution of a subpass. Each subpass **can** read from some attachments as *input attachments*, write to some as *color attachments* or *depth/stencil attachments*, and perform *multisample resolve operations* to *resolve attachments*. A subpass description **can** also include a set of *preserve attachments*, which are attachments that are not read or written by the subpass but whose contents **must** be preserved throughout the subpass.

A subpass *uses an attachment* if the attachment is a color, depth/stencil, resolve, depth/stencil resolve, fragment shading rate, or input attachment for that subpass (as determined by the **pColorAttachments**, **pDepthStencilAttachment**, **pResolveAttachments**, **VkSubpassDescriptionDepthStencilResolve::pDepthStencilResolveAttachment**, **VkFragmentShadingRateAttachmentInfoKHR::pFragmentShadingRateAttachment->attachment**, and **pInputAttachments** members of **VkSubpassDescription**, respectively). A subpass does not use an attachment if that attachment is preserved by the subpass. The *first use of an attachment* is in the lowest numbered subpass that uses that attachment. Similarly, the *last use of an attachment* is in the highest numbered subpass that uses that attachment.

The subpasses in a render pass all render to the same dimensions, and fragments for pixel (x,y,layer) in one subpass **can** only read attachment contents written by previous subpasses at that same (x,y,layer) location. For multi-pixel fragments, the pixel read from an input attachment is selected from the pixels covered by that fragment in an implementation-dependent manner. However, this selection **must** be made consistently for any fragment with the same shading rate for the lifetime of the **VkDevice**.

NOTE

By describing a complete set of subpasses in advance, render passes provide the implementation an opportunity to optimize the storage and transfer of attachment data between subpasses.

In practice, this means that subpasses with a simple framebuffer-space dependency

may be merged into a single tiled rendering pass, keeping the attachment data on-chip for the duration of a render pass instance. However, it is also quite common for a render pass to only contain a single subpass.

Subpass dependencies describe [execution and memory dependencies](#) between subpasses.

A *subpass dependency chain* is a sequence of subpass dependencies in a render pass, where the source subpass of each subpass dependency (after the first) equals the destination subpass of the previous dependency.

Execution of subpasses **may** overlap or execute out of order with regards to other subpasses, unless otherwise enforced by an execution dependency. Each subpass only respects [submission order](#) for commands recorded in the same subpass, and the [vkCmdBeginRenderPass](#) and [vkCmdEndRenderPass](#) commands that delimit the render pass - commands within other subpasses are not included. This affects most other [implicit ordering guarantees](#).

A render pass describes the structure of subpasses and attachments independent of any specific image views for the attachments. The specific image views that will be used for the attachments, and their dimensions, are specified in [VkFramebuffer](#) objects. Framebuffers are created with respect to a specific render pass that the framebuffer is compatible with (see [Render Pass Compatibility](#)). Collectively, a render pass and a framebuffer define the complete render target state for one or more subpasses as well as the algorithmic dependencies between the subpasses.

The various pipeline stages of the drawing commands for a given subpass **may** execute concurrently and/or out of order, both within and across drawing commands, whilst still respecting [pipeline order](#). However for a given (x,y,layer,sample) sample location, certain per-sample operations are performed in [rasterization order](#).

[VK_ATTACHMENT_UNUSED](#) is a constant indicating that a render pass attachment is not used.

```
#define VK_ATTACHMENT_UNUSED (~0U)
```

8.1. Render Pass Creation

To create a render pass, call:

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
VkResult vkCreateRenderPass(
    VkDevice                                device,
    const VkRenderPassCreateInfo*          pCreateInfo,
    const VkAllocationCallbacks*          pAllocator,
    VkRenderPass*                          pRenderPass);
```

- [device](#) is the logical device that creates the render pass.

- `pCreateInfo` is a pointer to a `VkRenderPassCreateInfo` structure describing the parameters of the render pass.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pRenderPass` is a pointer to a `VkRenderPass` handle in which the resulting render pass object is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateRenderPass` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateRenderPass-device-10000
`device` **must** support at least one queue family with the `VK_QUEUE_GRAPHICS_BIT` capability
- VUID-vkCreateRenderPass-device-05068
The number of render passes currently allocated from `device` plus 1 **must** be less than or equal to the total number of render passes requested via `VkDeviceObjectReservationCreateInfo::renderPassRequestCount` specified when `device` was created
- VUID-vkCreateRenderPass-subpasses-device-05089
The number of subpasses currently allocated from `device` across all `VkRenderPass` objects plus `pCreateInfo->subpassCount` **must** be less than or equal to the total number of subpasses requested via `VkDeviceObjectReservationCreateInfo::subpassDescriptionRequestCount` specified when `device` was created
- VUID-vkCreateRenderPass-attachments-device-05089
The number of attachments currently allocated from `device` across all `VkRenderPass` objects plus `pCreateInfo->attachmentCount` **must** be less than or equal to the total number of attachments requested via `VkDeviceObjectReservationCreateInfo::attachmentDescriptionRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateRenderPass-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkCreateRenderPass-pCreateInfo-parameter
`pCreateInfo` **must** be a valid pointer to a valid `VkRenderPassCreateInfo` structure
- VUID-vkCreateRenderPass-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateRenderPass-pRenderPass-parameter
`pRenderPass` **must** be a valid pointer to a `VkRenderPass` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkRenderPassCreateInfo` structure is defined as:

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
typedef struct VkRenderPassCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkRenderPassCreateFlags flags;
    uint32_t             attachmentCount;
    const VkAttachmentDescription* pAttachments;
    uint32_t             subpassCount;
    const VkSubpassDescription* pSubpasses;
    uint32_t             dependencyCount;
    const VkSubpassDependency* pDependencies;
} VkRenderPassCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `attachmentCount` is the number of attachments used by this render pass.
- `pAttachments` is a pointer to an array of `attachmentCount` `VkAttachmentDescription` structures describing the attachments used by the render pass.
- `subpassCount` is the number of subpasses to create.
- `pSubpasses` is a pointer to an array of `subpassCount` `VkSubpassDescription` structures describing each subpass.
- `dependencyCount` is the number of memory dependencies between pairs of subpasses.
- `pDependencies` is a pointer to an array of `dependencyCount` `VkSubpassDependency` structures describing dependencies between pairs of subpasses.

NOTE

Care should be taken to avoid a data race here; if any subpasses access attachments with overlapping memory locations, and one of those accesses is a write, a subpass dependency needs to be included between them.

Valid Usage

- VUID-VkRenderPassCreateInfo-attachment-00834

If the `attachment` member of any element of `pInputAttachments`, `pColorAttachments`, `pResolveAttachments` or `pDepthStencilAttachment`, or any element of `pPreserveAttachments` in any element of `pSubpasses` is not `VK_ATTACHMENT_UNUSED`, then it **must** be less than `attachmentCount`

- VUID-VkRenderPassCreateInfo-pAttachments-00836

For any member of `pAttachments` with a `loadOp` equal to `VK_ATTACHMENT_LOAD_OP_CLEAR`, the first use of that attachment **must** not specify a `layout` equal to `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`

- VUID-VkRenderPassCreateInfo-pAttachments-02511

For any member of `pAttachments` with a `stencilLoadOp` equal to `VK_ATTACHMENT_LOAD_OP_CLEAR`, the first use of that attachment **must** not specify a `layout` equal to `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`

- VUID-VkRenderPassCreateInfo-pAttachments-01566

For any member of `pAttachments` with a `loadOp` equal to `VK_ATTACHMENT_LOAD_OP_CLEAR`, the first use of that attachment **must** not specify a `layout` equal to `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`

- VUID-VkRenderPassCreateInfo-pAttachments-01567

For any member of `pAttachments` with a `stencilLoadOp` equal to `VK_ATTACHMENT_LOAD_OP_CLEAR`, the first use of that attachment **must** not specify a `layout` equal to `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`

- VUID-VkRenderPassCreateInfo-pNext-01926

If the `pNext` chain includes a `VkRenderPassInputAttachmentAspectCreateInfo` structure, the `subpass` member of each element of its `pAspectReferences` member **must** be less than `subpassCount`

- VUID-VkRenderPassCreateInfo-pNext-01927

If the `pNext` chain includes a `VkRenderPassInputAttachmentAspectCreateInfo` structure, the `inputAttachmentIndex` member of each element of its `pAspectReferences` member **must** be less than the value of `inputAttachmentCount` in the element of `pSubpasses` identified by its `subpass` member

- VUID-VkRenderPassCreateInfo-pNext-01963

If the `pNext` chain includes a `VkRenderPassInputAttachmentAspectCreateInfo` structure, for any element of the `pInputAttachments` member of any element of `pSubpasses` where the `attachment` member is not `VK_ATTACHMENT_UNUSED`, the `aspectMask` member of the corresponding element of `VkRenderPassInputAttachmentAspectCreateInfo::pAspectReferences` **must** only include aspects that are present in images of the format

specified by the element of `pAttachments` at `attachment`

- VUID-VkRenderPassCreateInfo-pNext-01928

If the `pNext` chain includes a `VkRenderPassMultiviewCreateInfo` structure, and its `subpassCount` member is not zero, that member **must** be equal to the value of `subpassCount`

- VUID-VkRenderPassCreateInfo-pNext-01929

If the `pNext` chain includes a `VkRenderPassMultiviewCreateInfo` structure, if its `dependencyCount` member is not zero, it **must** be equal to `dependencyCount`

- VUID-VkRenderPassCreateInfo-pNext-01930

If the `pNext` chain includes a `VkRenderPassMultiviewCreateInfo` structure, for each non-zero element of `pViewOffsets`, the `srcSubpass` and `dstSubpass` members of `pDependencies` at the same index **must** not be equal

- VUID-VkRenderPassCreateInfo-pNext-02512

If the `pNext` chain includes a `VkRenderPassMultiviewCreateInfo` structure, for any element of `pDependencies` with a `dependencyFlags` member that does not include `VK_DEPENDENCY_VIEW_LOCAL_BIT`, the corresponding element of the `pViewOffsets` member of that `VkRenderPassMultiviewCreateInfo` instance **must** be 0

- VUID-VkRenderPassCreateInfo-pNext-02513

If the `pNext` chain includes a `VkRenderPassMultiviewCreateInfo` structure, elements of its `pViewMasks` member **must** either all be 0, or all not be 0

- VUID-VkRenderPassCreateInfo-pNext-02514

If the `pNext` chain includes a `VkRenderPassMultiviewCreateInfo` structure, and each element of its `pViewMasks` member is 0, the `dependencyFlags` member of each element of `pDependencies` **must** not include `VK_DEPENDENCY_VIEW_LOCAL_BIT`

- VUID-VkRenderPassCreateInfo-pNext-02515

If the `pNext` chain includes a `VkRenderPassMultiviewCreateInfo` structure, and each element of its `pViewMasks` member is 0, its `correlationMaskCount` member **must** be 0

- VUID-VkRenderPassCreateInfo-pDependencies-00837

For any element of `pDependencies`, if the `srcSubpass` is not `VK_SUBPASS_EXTERNAL`, all stage flags included in the `srcStageMask` member of that dependency **must** be `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT` or a pipeline stage supported by the `pipeline` identified by the `pipelineBindPoint` member of the source subpass

- VUID-VkRenderPassCreateInfo-pDependencies-00838

For any element of `pDependencies`, if the `dstSubpass` is not `VK_SUBPASS_EXTERNAL`, all stage flags included in the `dstStageMask` member of that dependency **must** be `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT` or a pipeline stage supported by the `pipeline` identified by the `pipelineBindPoint` member of the destination subpass

- VUID-VkRenderPassCreateInfo-pDependencies-06866

For any element of `pDependencies`, if its `srcSubpass` is not `VK_SUBPASS_EXTERNAL`, it **must** be less than `subpassCount`

- VUID-VkRenderPassCreateInfo-pDependencies-06867

For any element of `pDependencies`, if its `dstSubpass` is not `VK_SUBPASS_EXTERNAL`, it **must** be less than `subpassCount`

- VUID-VkRenderPassCreateInfo-subpassCount-05050

`subpassCount` **must** be less than or equal to `maxRenderPassSubpasses`

- VUID-VkRenderPassCreateInfo-dependencyCount-05051
`dependencyCount` **must** be less than or equal to `maxRenderPassDependencies`
- VUID-VkRenderPassCreateInfo-attachmentCount-05052
`attachmentCount` **must** be less than or equal to `maxFramebufferAttachments`
- VUID-VkRenderPassCreateInfo-None-10915

If any subpass preserves an attachment, there **must** be a subpass dependency from a prior subpass which uses or preserves that attachment

Valid Usage (Implicit)

- VUID-VkRenderPassCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO`
- VUID-VkRenderPassCreateInfo-pNext-pNext
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkRenderPassInputAttachmentAspectCreateInfo` or `VkRenderPassMultiviewCreateInfo`
- VUID-VkRenderPassCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkRenderPassCreateInfo-flags-zerobitmask
`flags` **must** be `0`
- VUID-VkRenderPassCreateInfo-pAttachments-parameter
If `attachmentCount` is not `0`, `pAttachments` **must** be a valid pointer to an array of `attachmentCount` valid `VkAttachmentDescription` structures
- VUID-VkRenderPassCreateInfo-pSubpasses-parameter
`pSubpasses` **must** be a valid pointer to an array of `subpassCount` valid `VkSubpassDescription` structures
- VUID-VkRenderPassCreateInfo-pDependencies-parameter
If `dependencyCount` is not `0`, `pDependencies` **must** be a valid pointer to an array of `dependencyCount` valid `VkSubpassDependency` structures
- VUID-VkRenderPassCreateInfo-subpassCount-arraylength
`subpassCount` **must** be greater than `0`

Bits which **can** be set in `VkRenderPassCreateInfo::flags`, describing additional properties of the render pass, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkRenderPassCreateFlagBits {
} VkRenderPassCreateFlagBits;
```

NOTE

All bits for this type are defined by extensions, and none of those extensions are

enabled in this build of the specification.

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkRenderPassCreateFlags;
```

`VkRenderPassCreateFlags` is a bitmask type for setting a mask of zero or more `VkRenderPassCreateFlagBits`.

If the `VkRenderPassCreateInfo::pNext` chain includes a `VkRenderPassMultiviewCreateInfo` structure, then that structure includes an array of view masks, view offsets, and correlation masks for the render pass.

The `VkRenderPassMultiviewCreateInfo` structure is defined as:

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_1
typedef struct VkRenderPassMultiviewCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           subpassCount;
    const uint32_t*     pViewMasks;
    uint32_t           dependencyCount;
    const int32_t*      pViewOffsets;
    uint32_t           correlationMaskCount;
    const uint32_t*     pCorrelationMasks;
} VkRenderPassMultiviewCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `subpassCount` is zero or the number of subpasses in the render pass.
- `pViewMasks` is a pointer to an array of `subpassCount` view masks, where each mask is a bitfield of view indices describing which views rendering is broadcast to in each subpass, when multiview is enabled. If `subpassCount` is zero, each view mask is treated as zero.
- `dependencyCount` is zero or the number of dependencies in the render pass.
- `pViewOffsets` is a pointer to an array of `dependencyCount` view offsets, one for each dependency. If `dependencyCount` is zero, each dependency's view offset is treated as zero. Each view offset controls which views in the source subpass the views in the destination subpass depend on.
- `correlationMaskCount` is zero or the number of correlation masks.
- `pCorrelationMasks` is a pointer to an array of `correlationMaskCount` view masks indicating sets of

views that **may** be more efficient to render concurrently.

When a subpass uses a non-zero view mask, *multiview* functionality is considered to be enabled. Multiview is all-or-nothing for a render pass - that is, either all subpasses **must** have a non-zero view mask (though some subpasses **may** have only one view) or all **must** be zero. Multiview causes all drawing and clear commands in the subpass to behave as if they were broadcast to each view, where a view is represented by one layer of the framebuffer attachments. All draws and clears are broadcast to each *view index* whose bit is set in the view mask. The view index is provided in the *ViewIndex* shader input variable, and color, depth/stencil, and input attachments all read/write the layer of the framebuffer corresponding to the view index.

If the view mask is zero for all subpasses, multiview is considered to be disabled and all drawing commands execute normally, without this additional broadcasting.

Some implementations **may** not support multiview in conjunction with *geometry shaders* or *tessellation shaders*.

When multiview is enabled, the *VK_DEPENDENCY_VIEW_LOCAL_BIT* bit in a dependency **can** be used to express a view-local dependency, meaning that each view in the destination subpass depends on a single view in the source subpass. Unlike pipeline barriers, a subpass dependency **can** potentially have a different view mask in the source subpass and the destination subpass. If the dependency is view-local, then each view (*dstView*) in the destination subpass depends on the view *dstView + pViewOffsets[dependency]* in the source subpass. If there is not such a view in the source subpass, then this dependency does not affect that view in the destination subpass. If the dependency is not view-local, then all views in the destination subpass depend on all views in the source subpass, and the view offset is ignored. A non-zero view offset is not allowed in a self-dependency.

The elements of *pCorrelationMasks* are a set of masks of views indicating that views in the same mask **may** exhibit spatial coherency between the views, making it more efficient to render them concurrently. Correlation masks **must** not have a functional effect on the results of the multiview rendering.

When multiview is enabled, at the beginning of each subpass all non-render pass state is undefined. In particular, each time *vkCmdBeginRenderPass* or *vkCmdNextSubpass* is called the graphics pipeline **must** be bound, any relevant descriptor sets or vertex/index buffers **must** be bound, and any relevant dynamic state or push constants **must** be set before they are used.

Valid Usage

- VUID-VkRenderPassMultiviewCreateInfo-pCorrelationMasks-00841
Each view index **must** not be set in more than one element of *pCorrelationMasks*
- VUID-VkRenderPassMultiviewCreateInfo-multiview-06555
If the *multiview* feature is not enabled, each element of *pViewMasks* **must** be 0
- VUID-VkRenderPassMultiviewCreateInfo-pViewMasks-06697
The index of the most significant bit in each element of *pViewMasks* **must** be less than *maxMultiviewViewCount*

Valid Usage (Implicit)

- VUID-VkRenderPassMultiviewCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_MULTIVIEW_CREATE_INFO`
- VUID-VkRenderPassMultiviewCreateInfo-pViewMasks-parameter
If `subpassCount` is not 0, `pViewMasks` **must** be a valid pointer to an array of `subpassCount` `uint32_t` values
- VUID-VkRenderPassMultiviewCreateInfo-pViewOffsets-parameter
If `dependencyCount` is not 0, `pViewOffsets` **must** be a valid pointer to an array of `dependencyCount` `int32_t` values
- VUID-VkRenderPassMultiviewCreateInfo-pCorrelationMasks-parameter
If `correlationMaskCount` is not 0, `pCorrelationMasks` **must** be a valid pointer to an array of `correlationMaskCount` `uint32_t` values

The `VkAttachmentDescription` structure is defined as:

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
typedef struct VkAttachmentDescription {
    VkAttachmentDescriptionFlags    flags;
    VkFormat                       format;
    VkSampleCountFlagBits          samples;
    VkAttachmentLoadOp              loadOp;
    VkAttachmentStoreOp             storeOp;
    VkAttachmentLoadOp              stencilLoadOp;
    VkAttachmentStoreOp             stencilStoreOp;
    VkImageLayout                   initialLayout;
    VkImageLayout                   finalLayout;
} VkAttachmentDescription;
```

- `flags` is a bitmask of `VkAttachmentDescriptionFlagBits` specifying additional properties of the attachment.
- `format` is a `VkFormat` value specifying the format of the image view that will be used for the attachment.
- `samples` is a `VkSampleCountFlagBits` value specifying the number of samples of the image.
- `loadOp` is a `VkAttachmentLoadOp` value specifying how the contents of color and depth components of the attachment are treated at the beginning of the subpass where it is first used.
- `storeOp` is a `VkAttachmentStoreOp` value specifying how the contents of color and depth components of the attachment are treated at the end of the subpass where it is last used.
- `stencilLoadOp` is a `VkAttachmentLoadOp` value specifying how the contents of stencil components of the attachment are treated at the beginning of the subpass where it is first used.

- `stencilStoreOp` is a `VkAttachmentStoreOp` value specifying how the contents of stencil components of the attachment are treated at the end of the last subpass where it is used.
- `initialLayout` is the layout the attachment image subresource will be in when a render pass instance begins.
- `finalLayout` is the layout the attachment image subresource will be transitioned to when a render pass instance ends.

If the attachment uses a color format, then `loadOp` and `storeOp` are used, and `stencilLoadOp` and `stencilStoreOp` are ignored. If the format has depth and/or stencil components, `loadOp` and `storeOp` apply only to the depth data, while `stencilLoadOp` and `stencilStoreOp` define how the stencil data is handled. `loadOp` and `stencilLoadOp` define the `load operations` for the attachment. `storeOp` and `stencilStoreOp` define the `store operations` for the attachment. If an attachment is not used by any subpass, `loadOp`, `storeOp`, `stencilStoreOp`, and `stencilLoadOp` will be ignored for that attachment, and no load or store ops will be performed. However, any transition specified by `initialLayout` and `finalLayout` will still be executed.

If `flags` includes `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT`, then the attachment is treated as if it shares physical memory with another attachment in the same render pass. This information limits the ability of the implementation to reorder certain operations (like layout transitions and the `loadOp`) such that it is not improperly reordered against other uses of the same physical memory via a different attachment. This is described in more detail below.

If a render pass uses multiple attachments that alias the same device memory, those attachments **must** each include the `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT` bit in their attachment description flags. Attachments aliasing the same memory occurs in multiple ways:

- Multiple attachments being assigned the same image view as part of framebuffer creation.
- Attachments using distinct image views that correspond to the same image subresource of an image.
- Attachments using views of distinct image subresources which are bound to overlapping memory ranges.

NOTE

Render passes **must** include subpass dependencies (either directly or via a subpass dependency chain) between any two subpasses that operate on the same attachment or aliasing attachments and those subpass dependencies **must** include execution and memory dependencies separating uses of the aliases, if at least one of those subpasses writes to one of the aliases. These dependencies **must** not include the `VK_DEPENDENCY_BY_REGION_BIT` if the aliases are views of distinct image subresources which overlap in memory.

Multiple attachments that alias the same memory **must** not be used in a single subpass. A given attachment index **must** not be used multiple times in a single subpass, with one exception: two subpass attachments **can** use the same attachment index if at least one use is as an input attachment and neither use is as a resolve or preserve attachment. In other words, the same view **can** be used simultaneously as an input and color or depth/stencil attachment, but **must** not be used as multiple color or depth/stencil attachments nor as resolve or preserve attachments.

If a set of attachments alias each other, then all except the first to be used in the render pass **must** use an `initialLayout` of `VK_IMAGE_LAYOUT_UNDEFINED`, since the earlier uses of the other aliases make their contents undefined. Once an alias has been used and a different alias has been used after it, the first alias **must** not be used in any later subpasses. However, an application **can** assign the same image view to multiple aliasing attachment indices, which allows that image view to be used multiple times even if other aliases are used in between.

NOTE

Once an attachment needs the `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT` bit, there **should** be no additional cost of introducing additional aliases, and using these additional aliases **may** allow more efficient clearing of the attachments on multiple uses via `VK_ATTACHMENT_LOAD_OP_CLEAR`.

Valid Usage

- VUID-VkAttachmentDescription-format-06699
If `format` includes a color or depth component and `loadOp` is `VK_ATTACHMENT_LOAD_OP_LOAD`, then `initialLayout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED`
- VUID-VkAttachmentDescription-finalLayout-00843
`finalLayout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`
- VUID-VkAttachmentDescription-format-03280
If `format` is a color format, `initialLayout` **must** not be `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`
- VUID-VkAttachmentDescription-format-03281
If `format` is a depth/stencil format, `initialLayout` **must** not be `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`
- VUID-VkAttachmentDescription-format-03282
If `format` is a color format, `finalLayout` **must** not be `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`
- VUID-VkAttachmentDescription-format-03283
If `format` is a depth/stencil format, `finalLayout` **must** not be `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`
- VUID-VkAttachmentDescription-format-06487
If `format` is a color format, `initialLayout` **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`
- VUID-VkAttachmentDescription-format-06488
If `format` is a color format, `finalLayout` **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`
- VUID-VkAttachmentDescription-separateDepthStencilLayouts-03284
If the `separateDepthStencilLayouts` feature is not enabled, `initialLayout` **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`,

VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL,

- VUID-VkAttachmentDescription-separateDepthStencilLayouts-03285

If the `separateDepthStencilLayouts` feature is not enabled, `finalLayout` **must** not be
VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL,
VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL,

- VUID-VkAttachmentDescription-format-03286

If `format` is a color format, `initialLayout` **must** not be
VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL,
VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription-format-03287

If `format` is a color format, `finalLayout` **must** not be
VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL,
VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription-format-06906

If `format` is a depth/stencil format which includes both depth and stencil components,
`initialLayout` **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription-format-06907

If `format` is a depth/stencil format which includes both depth and stencil components,
`finalLayout` **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription-format-03290

If `format` is a depth/stencil format which includes only the depth component,
`initialLayout` **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription-format-03291

If `format` is a depth/stencil format which includes only the depth component, `finalLayout`
must not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription-synchronization2-06908

If the `synchronization2` feature is not enabled, `initialLayout` **must** not be
VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR

- VUID-VkAttachmentDescription-synchronization2-06909

If the `synchronization2` feature is not enabled, `finalLayout` **must** not be
VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR

- VUID-VkAttachmentDescription-samples-08745

`samples` **must** be a valid `VkSampleCountFlagBits` value that is set in
`imageCreateSampleCounts` (as defined in [Image Creation Limits](#)) for the given `format`

- VUID-VkAttachmentDescription-format-06698

format **must** not be `VK_FORMAT_UNDEFINED`

- VUID-VkAttachmentDescription-format-06700

If **format** includes a stencil component and **stencilLoadOp** is `VK_ATTACHMENT_LOAD_OP_LOAD`, then **initialLayout** **must** not be `VK_IMAGE_LAYOUT_UNDEFINED`

- VUID-VkAttachmentDescription-format-03292

If **format** is a depth/stencil format which includes only the stencil component, **initialLayout** **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`

- VUID-VkAttachmentDescription-format-03293

If **format** is a depth/stencil format which includes only the stencil component, **finalLayout** **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`

- VUID-VkAttachmentDescription-format-06242

If **format** is a depth/stencil format which includes both depth and stencil components, **initialLayout** **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`

- VUID-VkAttachmentDescription-format-06243

If **format** is a depth/stencil format which includes both depth and stencil components, **finalLayout** **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`

Valid Usage (Implicit)

- VUID-VkAttachmentDescription-flags-parameter

flags **must** be a valid combination of `VkAttachmentDescriptionFlagBits` values

- VUID-VkAttachmentDescription-format-parameter

format **must** be a valid `VkFormat` value

- VUID-VkAttachmentDescription-samples-parameter

samples **must** be a valid `VkSampleCountFlagBits` value

- VUID-VkAttachmentDescription-loadOp-parameter

loadOp **must** be a valid `VkAttachmentLoadOp` value

- VUID-VkAttachmentDescription-storeOp-parameter

storeOp **must** be a valid `VkAttachmentStoreOp` value

- VUID-VkAttachmentDescription-stencilLoadOp-parameter

stencilLoadOp **must** be a valid `VkAttachmentLoadOp` value

- VUID-VkAttachmentDescription-stencilStoreOp-parameter

stencilStoreOp **must** be a valid `VkAttachmentStoreOp` value

- VUID-VkAttachmentDescription-initialLayout-parameter

initialLayout **must** be a valid `VkImageLayout` value

- VUID-VkAttachmentDescription-finalLayout-parameter

finalLayout **must** be a valid `VkImageLayout` value

Bits which **can** be set in `VkAttachmentDescription::flags`, describing additional properties of the attachment, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkAttachmentDescriptionFlagBits {
    VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT = 0x00000001,
} VkAttachmentDescriptionFlagBits;
```

- `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT` specifies that the attachment aliases the same device memory as other attachments.

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkAttachmentDescriptionFlags;
```

`VkAttachmentDescriptionFlags` is a bitmask type for setting a mask of zero or more `VkAttachmentDescriptionFlagBits`.

The `VkRenderPassInputAttachmentAspectCreateInfo` structure is defined as:

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_1
typedef struct VkRenderPassInputAttachmentAspectCreateInfo {
    VkStructureType             sType;
    const void*                 pNext;
    uint32_t                    aspectReferenceCount;
    const VkInputAttachmentAspectReference* pAspectReferences;
} VkRenderPassInputAttachmentAspectCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `aspectReferenceCount` is the number of elements in the `pAspectReferences` array.
- `pAspectReferences` is a pointer to an array of `aspectReferenceCount` `VkInputAttachmentAspectReference` structures containing a mask describing which aspect(s) **can** be accessed for a given input attachment within a given subpass.

To specify which aspects of an input attachment **can** be read, add a `VkRenderPassInputAttachmentAspectCreateInfo` structure to the `pNext` chain of the `VkRenderPassCreateInfo` structure:

An application **can** access any aspect of an input attachment that does not have a specified aspect

mask in the `pAspectReferences` array. Otherwise, an application **must** not access aspect(s) of an input attachment other than those in its specified aspect mask.

Valid Usage (Implicit)

- VUID-VkRenderPassInputAttachmentAspectCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_INPUT_ATTACHMENT_ASPECT_CREATE_INFO`
- VUID-VkRenderPassInputAttachmentAspectCreateInfo-pAspectReferences-parameter
`pAspectReferences` **must** be a valid pointer to an array of `aspectReferenceCount` valid `VkInputAttachmentAspectReference` structures
- VUID-VkRenderPassInputAttachmentAspectCreateInfo-aspectReferenceCount-arraylength
`aspectReferenceCount` **must** be greater than 0

The `VkInputAttachmentAspectReference` structure is defined as:

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_1
typedef struct VkInputAttachmentAspectReference {
    uint32_t          subpass;
    uint32_t          inputAttachmentIndex;
    VkImageAspectFlags aspectMask;
} VkInputAttachmentAspectReference;
```

- `subpass` is an index into the `pSubpasses` array of the parent `VkRenderPassCreateInfo` structure.
- `inputAttachmentIndex` is an index into the `pInputAttachments` of the specified subpass.
- `aspectMask` is a mask of which aspect(s) **can** be accessed within the specified subpass.

This structure specifies an aspect mask for a specific input attachment of a specific subpass in the render pass.

`subpass` and `inputAttachmentIndex` index into the render pass as:

```
pCreateInfo->pSubpasses[subpass].pInputAttachments[inputAttachmentIndex]
```

Valid Usage

- VUID-VkInputAttachmentAspectReference-aspectMask-01964
`aspectMask` **must** not include `VK_IMAGE_ASPECT_METADATA_BIT`
- VUID-VkInputAttachmentAspectReference-aspectMask-02250
`aspectMask` **must** not include `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT` for any index *i*

Valid Usage (Implicit)

- VUID-VkInputAttachmentAspectReference-aspectMask-parameter **aspectMask** must be a valid combination of [VkImageAspectFlagBits](#) values
- VUID-VkInputAttachmentAspectReference-aspectMask-requiredbitmask **aspectMask** must not be 0

The [VkSubpassDescription](#) structure is defined as:

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
typedef struct VkSubpassDescription {
    VkSubpassDescriptionFlags    flags;
    VkPipelineBindPoint          pipelineBindPoint;
    uint32_t                     inputAttachmentCount;
    const VkAttachmentReference* pInputAttachments;
    uint32_t                     colorAttachmentCount;
    const VkAttachmentReference* pColorAttachments;
    const VkAttachmentReference* pResolveAttachments;
    const VkAttachmentReference* pDepthStencilAttachment;
    uint32_t                     preserveAttachmentCount;
    const uint32_t*              pPreserveAttachments;
} VkSubpassDescription;
```

- **flags** is a bitmask of [VkSubpassDescriptionFlagBits](#) specifying usage of the subpass.
- **pipelineBindPoint** is a [VkPipelineBindPoint](#) value specifying the pipeline type supported for this subpass.
- **inputAttachmentCount** is the number of input attachments.
- **pInputAttachments** is a pointer to an array of [VkAttachmentReference](#) structures defining the input attachments for this subpass and their layouts.
- **colorAttachmentCount** is the number of color attachments.
- **pColorAttachments** is a pointer to an array of **colorAttachmentCount** [VkAttachmentReference](#) structures defining the color attachments for this subpass and their layouts.
- **pResolveAttachments** is **NULL** or a pointer to an array of **colorAttachmentCount** [VkAttachmentReference](#) structures defining the resolve attachments for this subpass and their layouts.
- **pDepthStencilAttachment** is a pointer to a [VkAttachmentReference](#) structure specifying the depth/stencil attachment for this subpass and its layout.
- **preserveAttachmentCount** is the number of preserved attachments.
- **pPreserveAttachments** is a pointer to an array of **preserveAttachmentCount** render pass attachment

indices identifying attachments that are not used by this subpass, but whose contents **must** be preserved throughout the subpass.

Each element of the `pInputAttachments` array corresponds to an input attachment index in a fragment shader, i.e. if a shader declares an image variable decorated with a `InputAttachmentIndex` value of `X`, then it uses the attachment provided in `pInputAttachments[X]`. Input attachments **must** also be bound to the pipeline in a descriptor set. If the `attachment` member of any element of `pInputAttachments` is `VK_ATTACHMENT_UNUSED`, the application **must** not read from the corresponding input attachment index. Fragment shaders **can** use subpass input variables to access the contents of an input attachment at the fragment's (x_f, y_f) framebuffer coordinates and layer.

Each element of the `pColorAttachments` array corresponds to an output location in the shader, i.e. if the shader declares an output variable decorated with a `Location` value of `X`, then it uses the attachment provided in `pColorAttachments[X]`. If the `attachment` member of any element of `pColorAttachments` is `VK_ATTACHMENT_UNUSED`, or if `Color Write Enable` has been disabled for the corresponding attachment index, then writes to the corresponding location by a fragment shader are discarded.

If `pResolveAttachments` is not `NULL`, each of its elements corresponds to a color attachment (the element in `pColorAttachments` at the same index), and a `multisample resolve operation` is defined for each attachment unless the resolve attachment index is `VK_ATTACHMENT_UNUSED`.

Similarly, if `VkSubpassDescriptionDepthStencilResolve::pDepthStencilResolveAttachment` is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED`, it corresponds to the depth/stencil attachment in `pDepthStencilAttachment`, and `multisample resolve operation` for depth and stencil are defined by `VkSubpassDescriptionDepthStencilResolve::depthResolveMode` and `VkSubpassDescriptionDepthStencilResolve::stencilResolveMode`, respectively. If `VkSubpassDescriptionDepthStencilResolve::depthResolveMode` is `VK_RESOLVE_MODE_NONE` or the `pDepthStencilResolveAttachment` does not have a depth aspect, no resolve operation is performed for the depth attachment. If `VkSubpassDescriptionDepthStencilResolve::stencilResolveMode` is `VK_RESOLVE_MODE_NONE` or the `pDepthStencilResolveAttachment` does not have a stencil aspect, no resolve operation is performed for the stencil attachment.

If the image subresource range referenced by the depth/stencil attachment is created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT`, then the `multisample resolve operation` uses the sample locations state specified in the `sampleLocationsInfo` member of the element of the `VkRenderPassSampleLocationsBeginInfoEXT::pPostSubpassSampleLocations` for the subpass.

If `pDepthStencilAttachment` is `NULL`, or if its attachment index is `VK_ATTACHMENT_UNUSED`, it indicates that no depth/stencil attachment will be used in the subpass.

The contents of an attachment within the render area become undefined at the start of a subpass `S` if all of the following conditions are true:

- The attachment is used as a color, depth/stencil, or resolve attachment in any subpass in the render pass.
- There is a subpass `S1` that uses or preserves the attachment, and a subpass dependency from `S1` to `S`.

- The attachment is not used or preserved in subpass **S**.

For color attachments, this operation will be performed in the `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT` pipeline stage, with any image accesses performed via `VK_ACCESS_INPUT_ATTACHMENT_READ_BIT`, `VK_ACCESS_COLOR_ATTACHMENT_READ_BIT`, and `VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT`. For depth/stencil attachments, this operation **may** be performed in either the `VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT` or `VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT` pipeline stage, with any image accesses performed via `VK_ACCESS_INPUT_ATTACHMENT_READ_BIT`, `VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT`, and `VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`.

Once the contents of an attachment become undefined in subpass **S**, they remain undefined for subpasses in subpass dependency chains starting with subpass **S** until they are written again. However, they remain valid for subpasses in other subpass dependency chains starting with subpass **S**₁ if all subpasses in each chain use or preserve the attachment.

NOTE

If a subpass has multiple dependency chains where some of the chains preserve the attachment, and others do not, the contents of the attachment are undefined for that subpass.

Valid Usage

- VUID-VkSubpassDescription-attachment-06912

If the `attachment` member of an element of `pInputAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL`

- VUID-VkSubpassDescription-attachment-06913

If the `attachment` member of an element of `pColorAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`

- VUID-VkSubpassDescription-attachment-06914

If the `attachment` member of an element of `pResolveAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`

- VUID-VkSubpassDescription-attachment-06915

If the `attachment` member of `pDepthStencilAttachment` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`

- VUID-VkSubpassDescription-attachment-06916

If the `attachment` member of an element of `pColorAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`

- VUID-VkSubpassDescription-attachment-06917

If the `attachment` member of an element of `pResolveAttachments` is not

VK_ATTACHMENT_UNUSED, its layout member **must** not be
VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL or
VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL

- VUID-VkSubpassDescription-attachment-06918

If the `attachment` member of an element of `pInputAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`

- VUID-VkSubpassDescription-attachment-06919

If the `attachment` member of an element of `pColorAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`

- VUID-VkSubpassDescription-attachment-06920

If the `attachment` member of an element of `pResolveAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`

- VUID-VkSubpassDescription-attachment-06921

If the `attachment` member of an element of `pInputAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR`

- VUID-VkSubpassDescription-attachment-06922

If the `attachment` member of an element of `pColorAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`

- VUID-VkSubpassDescription-attachment-06923

If the `attachment` member of an element of `pResolveAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`

- VUID-VkSubpassDescription-pipelineBindPoint-04952

`pipelineBindPoint` **must** be `VK_PIPELINE_BIND_POINT_GRAPHICS`

- VUID-VkSubpassDescription-colorAttachmentCount-00845

`colorAttachmentCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxColorAttachments`

- VUID-VkSubpassDescription-loadOp-00846

If the first use of an attachment in this render pass is as an input attachment, and the attachment is not also used as a color or depth/stencil attachment in the same subpass, then `loadOp` **must** not be `VK_ATTACHMENT_LOAD_OP_CLEAR`

- VUID-VkSubpassDescription-pResolveAttachments-00847

If `pResolveAttachments` is not `NULL`, for each resolve attachment that is not `VK_ATTACHMENT_UNUSED`, the corresponding color attachment **must** not be `VK_ATTACHMENT_UNUSED`

- VUID-VkSubpassDescription-pResolveAttachments-00848

If `pResolveAttachments` is not `NULL`, for each resolve attachment that is not

VK_ATTACHMENT_UNUSED, the corresponding color attachment **must** not have a sample count of VK_SAMPLE_COUNT_1_BIT

- VUID-VkSubpassDescription-pResolveAttachments-00849
If `pResolveAttachments` is not `NULL`, each resolve attachment that is not `VK_ATTACHMENT_UNUSED` **must** have a sample count of `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkSubpassDescription-pResolveAttachments-00850
If `pResolveAttachments` is not `NULL`, each resolve attachment that is not `VK_ATTACHMENT_UNUSED` **must** have the same `VkFormat` as its corresponding color attachment
- VUID-VkSubpassDescription-pColorAttachments-09430
All attachments in `pColorAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have the same sample count
- VUID-VkSubpassDescription-pInputAttachments-02647
All attachments in `pInputAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have image formats whose `potential format features` contain at least `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` or `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkSubpassDescription-pColorAttachments-02648
All attachments in `pColorAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have image formats whose `potential format features` contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`
- VUID-VkSubpassDescription-pResolveAttachments-02649
All attachments in `pResolveAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have image formats whose `potential format features` contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`
- VUID-VkSubpassDescription-pDepthStencilAttachment-02650
If `pDepthStencilAttachment` is not `NULL` and the attachment is not `VK_ATTACHMENT_UNUSED` then it **must** have an image format whose `potential format features` contain `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkSubpassDescription-pDepthStencilAttachment-01418
If `pDepthStencilAttachment` is not `VK_ATTACHMENT_UNUSED` and any attachments in `pColorAttachments` are not `VK_ATTACHMENT_UNUSED`, they **must** have the same sample count
- VUID-VkSubpassDescription-attachment-00853
Each element of `pPreserveAttachments` **must** not be `VK_ATTACHMENT_UNUSED`
- VUID-VkSubpassDescription-pPreserveAttachments-00854
Each element of `pPreserveAttachments` **must** not also be an element of any other member of the subpass description
- VUID-VkSubpassDescription-layout-02519
If any attachment is used by more than one `VkAttachmentReference` member, then each use **must** use the same `layout`
- VUID-VkSubpassDescription-pDepthStencilAttachment-04438
`pDepthStencilAttachment` and `pColorAttachments` **must** not contain references to the same attachment
- VUID-VkSubpassDescription-inputAttachmentCount-05053

`inputAttachmentCount` **must** be less than or equal to `maxSubpassInputAttachments`

- VUID-VkSubpassDescription-preserveAttachmentCount-05054

`preserveAttachmentCount` **must** be less than or equal to `maxSubpassPreserveAttachments`

Valid Usage (Implicit)

- VUID-VkSubpassDescription-flags-zeroBitmask
`flags` **must** be 0
- VUID-VkSubpassDescription-pipelineBindPoint-parameter
`pipelineBindPoint` **must** be a valid `VkPipelineBindPoint` value
- VUID-VkSubpassDescription-pInputAttachments-parameter
If `inputAttachmentCount` is not 0, `pInputAttachments` **must** be a valid pointer to an array of `inputAttachmentCount` valid `VkAttachmentReference` structures
- VUID-VkSubpassDescription-pColorAttachments-parameter
If `colorAttachmentCount` is not 0, `pColorAttachments` **must** be a valid pointer to an array of `colorAttachmentCount` valid `VkAttachmentReference` structures
- VUID-VkSubpassDescription-pResolveAttachments-parameter
If `colorAttachmentCount` is not 0, and `pResolveAttachments` is not NULL, `pResolveAttachments` **must** be a valid pointer to an array of `colorAttachmentCount` valid `VkAttachmentReference` structures
- VUID-VkSubpassDescription-pDepthStencilAttachment-parameter
If `pDepthStencilAttachment` is not NULL, `pDepthStencilAttachment` **must** be a valid pointer to a valid `VkAttachmentReference` structure
- VUID-VkSubpassDescription-pPreserveAttachments-parameter
If `preserveAttachmentCount` is not 0, `pPreserveAttachments` **must** be a valid pointer to an array of `preserveAttachmentCount` `uint32_t` values

Bits which **can** be set in `VkSubpassDescription::flags`, specifying usage of the subpass, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSubpassDescriptionFlagBits {
} VkSubpassDescriptionFlagBits;
```

NOTE

All bits for this type are defined by extensions, and none of those extensions are enabled in this build of the specification.

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkSubpassDescriptionFlags;
```

`VkSubpassDescriptionFlags` is a bitmask type for setting a mask of zero or more `VkSubpassDescriptionFlagBits`.

The `VkAttachmentReference` structure is defined as:

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
typedef struct VkAttachmentReference {
    uint32_t      attachment;
    VkImageLayout layout;
} VkAttachmentReference;
```

- `attachment` is either an integer value identifying an attachment at the corresponding index in `VkRenderPassCreateInfo::pAttachments`, or `VK_ATTACHMENT_UNUSED` to signify that this attachment is not used.
- `layout` is a `VkImageLayout` value specifying the layout the attachment uses during the subpass.

Valid Usage

- VUID-VkAttachmentReference-layout-03077
If `attachment` is not `VK_ATTACHMENT_UNUSED`, `layout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED`, `VK_IMAGE_LAYOUT_PREINITIALIZED`, or `VK_IMAGE_LAYOUT_PRESENT_SRC_KHR`
- VUID-VkAttachmentReference-separateDepthStencilLayouts-03313
If the `separateDepthStencilLayouts` feature is not enabled, and `attachment` is not `VK_ATTACHMENT_UNUSED`, `layout` **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`,
- VUID-VkAttachmentReference-synchronization2-06910
If the `synchronization2` feature is not enabled, `layout` **must** not be `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR` or `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`

Valid Usage (Implicit)

- VUID-VkAttachmentReference-layout-parameter
`layout` **must** be a valid `VkImageLayout` value

`VK_SUBPASS_EXTERNAL` is a special subpass index value expanding synchronization scope outside a subpass. It is described in more detail by [VkSubpassDependency](#).

```
#define VK_SUBPASS_EXTERNAL (~0U)
```

The `VkSubpassDependency` structure is defined as:

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
typedef struct VkSubpassDependency {
    uint32_t          srcSubpass;
    uint32_t          dstSubpass;
    VkPipelineStageFlags srcStageMask;
    VkPipelineStageFlags dstStageMask;
    VkAccessFlags      srcAccessMask;
    VkAccessFlags      dstAccessMask;
    VkDependencyFlags  dependencyFlags;
} VkSubpassDependency;
```

- `srcSubpass` is the subpass index of the first subpass in the dependency, or `VK_SUBPASS_EXTERNAL`.
- `dstSubpass` is the subpass index of the second subpass in the dependency, or `VK_SUBPASS_EXTERNAL`.
- `srcStageMask` is a bitmask of `VkPipelineStageFlagBits` specifying the [source stage mask](#). If set to `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT`, it is equivalent to setting it to `VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT`.
- `dstStageMask` is a bitmask of `VkPipelineStageFlagBits` specifying the [destination stage mask](#). If set to `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT`, it is equivalent to setting it to `VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT`.
- `srcAccessMask` is a bitmask of `VkAccessFlagBits` specifying a [source access mask](#).
- `dstAccessMask` is a bitmask of `VkAccessFlagBits` specifying a [destination access mask](#).
- `dependencyFlags` is a bitmask of `VkDependencyFlagBits`.

If `srcSubpass` is equal to `dstSubpass` then the `VkSubpassDependency` does not directly define a [dependency](#). Instead, it enables pipeline barriers to be used in a render pass instance within the identified subpass, where the scopes of one pipeline barrier **must** be a subset of those described by one subpass dependency. Subpass dependencies specified in this way that include [framebuffer-space stages](#) in the `srcStageMask` **must** only include [framebuffer-space stages](#) in `dstStageMask`, and **must** include `VK_DEPENDENCY_BY_REGION_BIT`. When a subpass dependency is specified in this way for a subpass that has more than one view in its view mask, its `dependencyFlags` **must** include `VK_DEPENDENCY_VIEW_LOCAL_BIT`.

If `srcSubpass` and `dstSubpass` are not equal, when a render pass instance which includes a subpass dependency is submitted to a queue, it defines a [dependency](#) between the subpasses identified by `srcSubpass` and `dstSubpass`.

If `srcSubpass` is equal to `VK_SUBPASS_EXTERNAL`, the first [synchronization scope](#) includes commands that occur earlier in [submission order](#) than the `vkCmdBeginRenderPass` used to begin the render pass instance. Otherwise, the first set of commands includes all commands submitted as part of the

subpass instance identified by `srcSubpass` and any `load`, `store`, or `multisample resolve` operations on attachments used in `srcSubpass`. In either case, the first synchronization scope is limited to operations on the pipeline stages determined by the `source stage mask` specified by `srcStageMask`.

If `dstSubpass` is equal to `VK_SUBPASS_EXTERNAL`, the second `synchronization scope` includes commands that occur later in `submission order` than the `vkCmdEndRenderPass` used to end the render pass instance. Otherwise, the second set of commands includes all commands submitted as part of the subpass instance identified by `dstSubpass` and any `load`, `store`, and `multisample resolve` operations on attachments used in `dstSubpass`. In either case, the second synchronization scope is limited to operations on the pipeline stages determined by the `destination stage mask` specified by `dstStageMask`.

The first `access scope` is limited to accesses in the pipeline stages determined by the `source stage mask` specified by `srcStageMask`. It is also limited to access types in the `source access mask` specified by `srcAccessMask`.

The second `access scope` is limited to accesses in the pipeline stages determined by the `destination stage mask` specified by `dstStageMask`. It is also limited to access types in the `destination access mask` specified by `dstAccessMask`.

The `availability and visibility operations` defined by a subpass dependency affect the execution of `image layout transitions` within the render pass.

For non-attachment resources, the memory dependency expressed by subpass dependency is nearly identical to that of a `VkMemoryBarrier` (with matching `srcAccessMask` and `dstAccessMask` parameters) submitted as a part of a `vkCmdPipelineBarrier` (with matching `srcStageMask` and `dstStageMask` parameters). The only difference being that its scopes are limited to the identified subpasses rather than potentially affecting everything before and after.

NOTE

For attachments however, subpass dependencies work more like a `VkImageMemoryBarrier` defined similarly to the `VkMemoryBarrier` above, the queue family indices set to `VK_QUEUE_FAMILY_IGNORED`, and layouts as follows:

- The equivalent to `oldLayout` is the attachment's layout according to the subpass description for `srcSubpass`.
- The equivalent to `newLayout` is the attachment's layout according to the subpass description for `dstSubpass`.

Valid Usage

- VUID-VkSubpassDependency-srcStageMask-04090

If the `geometryShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`

- VUID-VkSubpassDependency-srcStageMask-04091

If the `tessellationShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`

- VUID-VkSubpassDependency-srcStageMask-07319
If the `attachmentFragmentShadingRate` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-VkSubpassDependency-srcStageMask-03937
If the `synchronization2` feature is not enabled, `srcStageMask` **must** not be `0`
- VUID-VkSubpassDependency-dstStageMask-04090
If the `geometryShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-VkSubpassDependency-dstStageMask-04091
If the `tessellationShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-VkSubpassDependency-dstStageMask-07319
If the `attachmentFragmentShadingRate` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-VkSubpassDependency-dstStageMask-03937
If the `synchronization2` feature is not enabled, `dstStageMask` **must** not be `0`
- VUID-VkSubpassDependency-srcSubpass-00864
`srcSubpass` **must** be less than or equal to `dstSubpass`, unless one of them is `VK_SUBPASS_EXTERNAL`, to avoid cyclic dependencies and ensure a valid execution order
- VUID-VkSubpassDependency-srcSubpass-00865
`srcSubpass` and `dstSubpass` **must** not both be equal to `VK_SUBPASS_EXTERNAL`
- VUID-VkSubpassDependency-srcSubpass-06809
If `srcSubpass` is equal to `dstSubpass` and `srcStageMask` includes a `framebuffer-space stage`, `dstStageMask` **must** only contain `framebuffer-space stages`
- VUID-VkSubpassDependency-srcAccessMask-00868
Any access flag included in `srcAccessMask` **must** be supported by one of the pipeline stages in `srcStageMask`, as specified in the `table of supported access types`
- VUID-VkSubpassDependency-dstAccessMask-00869
Any access flag included in `dstAccessMask` **must** be supported by one of the pipeline stages in `dstStageMask`, as specified in the `table of supported access types`
- VUID-VkSubpassDependency-srcSubpass-02243
If `srcSubpass` equals `dstSubpass`, and `srcStageMask` and `dstStageMask` both include a `framebuffer-space stage`, then `dependencyFlags` **must** include `VK_DEPENDENCY_BY_REGION_BIT`
- VUID-VkSubpassDependency-dependencyFlags-02520
If `dependencyFlags` includes `VK_DEPENDENCY_VIEW_LOCAL_BIT`, `srcSubpass` **must** not be equal to `VK_SUBPASS_EXTERNAL`
- VUID-VkSubpassDependency-dependencyFlags-02521
If `dependencyFlags` includes `VK_DEPENDENCY_VIEW_LOCAL_BIT`, `dstSubpass` **must** not be equal to `VK_SUBPASS_EXTERNAL`
- VUID-VkSubpassDependency-srcSubpass-00872
If `srcSubpass` equals `dstSubpass` and that subpass has more than one bit set in the view

mask, then `dependencyFlags` **must** include `VK_DEPENDENCY_VIEW_LOCAL_BIT`

Valid Usage (Implicit)

- `VUID-VkSubpassDependency-srcStageMask-parameter`
`srcStageMask` **must** be a valid combination of `VkPipelineStageFlagBits` values
- `VUID-VkSubpassDependency-dstStageMask-parameter`
`dstStageMask` **must** be a valid combination of `VkPipelineStageFlagBits` values
- `VUID-VkSubpassDependency-srcAccessMask-parameter`
`srcAccessMask` **must** be a valid combination of `VkAccessFlagBits` values
- `VUID-VkSubpassDependency-dstAccessMask-parameter`
`dstAccessMask` **must** be a valid combination of `VkAccessFlagBits` values
- `VUID-VkSubpassDependency-dependencyFlags-parameter`
`dependencyFlags` **must** be a valid combination of `VkDependencyFlagBits` values

When multiview is enabled, the execution of the multiple views of one subpass **may** not occur simultaneously or even back-to-back, and rather **may** be interleaved with the execution of other subpasses. The load and store operations apply to attachments on a per-view basis. For example, an attachment using `VK_ATTACHMENT_LOAD_OP_CLEAR` will have each view cleared on first use, but the first use of one view may be temporally distant from the first use of another view.

NOTE

A good mental model for multiview is to think of a multiview subpass as if it were a collection of individual (per-view) subpasses that are logically grouped together and described as a single multiview subpass in the API. Similarly, a multiview attachment can be thought of like several individual attachments that happen to be layers in a single image. A view-local dependency between two multiview subpasses acts like a set of one-to-one dependencies between corresponding pairs of per-view subpasses. A view-global dependency between two multiview subpasses acts like a set of $N \times M$ dependencies between all pairs of per-view subpasses in the source and destination. Thus, it is a more compact representation which also makes clear the commonality and reuse that is present between views in a subpass. This interpretation motivates the answers to questions like “when does the load op apply” - it is on the first use of each view of an attachment, as if each view was a separate attachment.

The content of each view follows the description in [attachment content behavior](#). In particular, if an attachment is preserved, all views within the attachment are preserved.

If there is no subpass dependency from `VK_SUBPASS_EXTERNAL` to the first subpass that uses an attachment, then an implicit subpass dependency exists from `VK_SUBPASS_EXTERNAL` to the first subpass it is used in. The implicit subpass dependency only exists if there exists an automatic layout transition away from `initialLayout`. The subpass dependency operates as if defined with the following parameters:

```

VkSubpassDependency implicitDependency = {
    .srcSubpass = VK_SUBPASS_EXTERNAL,
    .dstSubpass = firstSubpass, // First subpass attachment is used in
    .srcStageMask = VK_PIPELINE_STAGE_NONE,
    .dstStageMask = VK_PIPELINE_STAGE_ALL_COMMANDS_BIT,
    .srcAccessMask = 0,
    .dstAccessMask = VK_ACCESS_INPUT_ATTACHMENT_READ_BIT |
                    VK_ACCESS_COLOR_ATTACHMENT_READ_BIT |
                    VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT |
                    VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT |
                    VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT,
    .dependencyFlags = 0
};

```

Similarly, if there is no subpass dependency from the last subpass that uses an attachment to `VK_SUBPASS_EXTERNAL`, then an implicit subpass dependency exists from the last subpass it is used in to `VK_SUBPASS_EXTERNAL`. The implicit subpass dependency only exists if there exists an automatic layout transition into `finalLayout`. The subpass dependency operates as if defined with the following parameters:

```

VkSubpassDependency implicitDependency = {
    .srcSubpass = lastSubpass, // Last subpass attachment is used in
    .dstSubpass = VK_SUBPASS_EXTERNAL,
    .srcStageMask = VK_PIPELINE_STAGE_ALL_COMMANDS_BIT,
    .dstStageMask = VK_PIPELINE_STAGE_NONE,
    .srcAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT |
                    VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT,
    .dstAccessMask = 0,
    .dependencyFlags = 0
};

```

As subpasses **may** overlap or execute out of order with regards to other subpasses unless a subpass dependency chain describes otherwise, the layout transitions required between subpasses **cannot** be known to an application. Instead, an application provides the layout that each attachment **must** be in at the start and end of a render pass, and the layout it **must** be in during each subpass it is used in. The implementation then **must** execute layout transitions between subpasses in order to guarantee that the images are in the layouts required by each subpass, and in the final layout at the end of the render pass.

Automatic layout transitions apply to the entire image subresource attached to the framebuffer. If multiview is not enabled and the attachment is a view of a 1D or 2D image, the automatic layout transitions apply to the number of layers specified by `VkFramebufferCreateInfo::layers`. If multiview is enabled and the attachment is a view of a 1D or 2D image, the automatic layout transitions apply to the layers corresponding to views which are used by some subpass in the render pass, even if that subpass does not reference the given attachment. If the attachment view is a 2D or 2D array view of a 3D image, even if the attachment view only refers to a subset of the slices of the selected mip level of the 3D image, automatic layout transitions apply to the entire

subresource referenced which is the entire mip level in this case.

Automatic layout transitions away from the layout used in a subpass happen-after the availability operations for all dependencies with that subpass as the `srcSubpass`.

Automatic layout transitions into the layout used in a subpass happen-before the visibility operations for all dependencies with that subpass as the `dstSubpass`.

Automatic layout transitions away from `initialLayout` happen-after the availability operations for all dependencies with a `srcSubpass` equal to `VK_SUBPASS_EXTERNAL`, where `dstSubpass` uses the attachment that will be transitioned. For attachments created with `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT`, automatic layout transitions away from `initialLayout` happen-after the availability operations for all dependencies with a `srcSubpass` equal to `VK_SUBPASS_EXTERNAL`, where `dstSubpass` uses any aliased attachment.

Automatic layout transitions into `finalLayout` happen-before the visibility operations for all dependencies with a `dstSubpass` equal to `VK_SUBPASS_EXTERNAL`, where `srcSubpass` uses the attachment that will be transitioned. For attachments created with `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT`, automatic layout transitions into `finalLayout` happen-before the visibility operations for all dependencies with a `dstSubpass` equal to `VK_SUBPASS_EXTERNAL`, where `srcSubpass` uses any aliased attachment.

The image layout of the depth aspect of a depth/stencil attachment referring to an image created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` is dependent on the last sample locations used to render to the attachment, thus automatic layout transitions use the sample locations state specified in `VkRenderPassSampleLocationsBeginInfoEXT`.

Automatic layout transitions of an attachment referring to a depth/stencil image created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` use the sample locations the image subresource range referenced by the attachment was last rendered with. If the current render pass does not use the attachment as a depth/stencil attachment in any subpass that happens-before, the automatic layout transition uses the sample locations state specified in the `sampleLocationsInfo` member of the element of the `VkRenderPassSampleLocationsBeginInfoEXT::pAttachmentInitialSampleLocations` array for which the `attachmentIndex` member equals the attachment index of the attachment, if one is specified. Otherwise, the automatic layout transition uses the sample locations state specified in the `sampleLocationsInfo` member of the element of the `VkRenderPassSampleLocationsBeginInfoEXT::pPostSubpassSampleLocations` array for which the `subpassIndex` member equals the index of the subpass that last used the attachment as a depth/stencil attachment, if one is specified.

If no sample locations state has been specified for an automatic layout transition performed on an attachment referring to a depth/stencil image created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` the contents of the depth aspect of the depth/stencil attachment become undefined as if the layout of the attachment was transitioned from the `VK_IMAGE_LAYOUT_UNDEFINED` layout.

If two subpasses use the same attachment, and both subpasses use the attachment in a read-only layout, no subpass dependency needs to be specified between those subpasses. If an implementation treats those layouts separately, it **must** insert an implicit subpass dependency

between those subpasses to separate the uses in each layout. The subpass dependency operates as if defined with the following parameters:

```
// Used for input attachments
VkPipelineStageFlags inputAttachmentStages = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
VkAccessFlags inputAttachmentDstAccess = VK_ACCESS_INPUT_ATTACHMENT_READ_BIT;

// Used for depth/stencil attachments
VkPipelineStageFlags depthStencilAttachmentStages =
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT |
VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT;
VkAccessFlags depthStencilAttachmentDstAccess =
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT;

VkSubpassDependency implicitDependency = {
    .srcSubpass = firstSubpass;
    .dstSubpass = secondSubpass;
    .srcStageMask = inputAttachmentStages | depthStencilAttachmentStages;
    .dstStageMask = inputAttachmentStages | depthStencilAttachmentStages;
    .srcAccessMask = 0;
    .dstAccessMask = inputAttachmentDstAccess | depthStencilAttachmentDstAccess;
    .dependencyFlags = 0;
};
```

A more extensible version of render pass creation is also defined below.

To create a render pass, call:

```
// Provided by VK_VERSION_1_2
VkResult vkCreateRenderPass2(
    VkDevice                                device,
    const VkRenderPassCreateInfo2*          pCreateInfo,
    const VkAllocationCallbacks*            pAllocator,
    VkRenderPass*                           pRenderPass);
```

- **device** is the logical device that creates the render pass.
- **pCreateInfo** is a pointer to a [VkRenderPassCreateInfo2](#) structure describing the parameters of the render pass.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.
- **pRenderPass** is a pointer to a [VkRenderPass](#) handle in which the resulting render pass object is returned.

This command is functionally identical to [vkCreateRenderPass](#), but includes extensible substructures that include **sType** and **pNext** parameters, allowing them to be more easily extended.

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is **VK_TRUE**, **vkCreateRenderPass2** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage

- VUID-vkCreateRenderPass2-device-10001
device **must** support at least one queue family with the `VK_QUEUE_GRAPHICS_BIT` capability
- VUID-vkCreateRenderPass2-device-05068
The number of render passes currently allocated from **device** plus 1 **must** be less than or equal to the total number of render passes requested via `VkDeviceObjectReservationCreateInfo::renderPassRequestCount` specified when **device** was created
- VUID-vkCreateRenderPass2-subpasses-device-05089
The number of subpasses currently allocated from **device** across all `VkRenderPass` objects plus `pCreateInfo->subpassCount` **must** be less than or equal to the total number of subpasses requested via `VkDeviceObjectReservationCreateInfo::subpassDescriptionRequestCount` specified when **device** was created
- VUID-vkCreateRenderPass2-attachments-device-05089
The number of attachments currently allocated from **device** across all `VkRenderPass` objects plus `pCreateInfo->attachmentCount` **must** be less than or equal to the total number of attachments requested via `VkDeviceObjectReservationCreateInfo::attachmentDescriptionRequestCount` specified when **device** was created

Valid Usage (Implicit)

- VUID-vkCreateRenderPass2-device-parameter
device **must** be a valid `VkDevice` handle
- VUID-vkCreateRenderPass2-pCreateInfo-parameter
pCreateInfo **must** be a valid pointer to a valid `VkRenderPassCreateInfo2` structure
- VUID-vkCreateRenderPass2-pAllocator-null
pAllocator **must** be `NULL`
- VUID-vkCreateRenderPass2-pRenderPass-parameter
pRenderPass **must** be a valid pointer to a `VkRenderPass` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkRenderPassCreateInfo2` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkRenderPassCreateInfo2 {
    VkStructureType      sType;
    const void*          pNext;
    VkRenderPassCreateFlags flags;
    uint32_t             attachmentCount;
    const VkAttachmentDescription2* pAttachments;
    uint32_t             subpassCount;
    const VkSubpassDescription2* pSubpasses;
    uint32_t             dependencyCount;
    const VkSubpassDependency2* pDependencies;
    uint32_t             correlatedViewMaskCount;
    const uint32_t*       pCorrelatedViewMasks;
} VkRenderPassCreateInfo2;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `attachmentCount` is the number of attachments used by this render pass.
- `pAttachments` is a pointer to an array of `attachmentCount` `VkAttachmentDescription2` structures describing the attachments used by the render pass.
- `subpassCount` is the number of subpasses to create.
- `pSubpasses` is a pointer to an array of `subpassCount` `VkSubpassDescription2` structures describing each subpass.
- `dependencyCount` is the number of dependencies between pairs of subpasses.
- `pDependencies` is a pointer to an array of `dependencyCount` `VkSubpassDependency2` structures describing dependencies between pairs of subpasses.
- `correlatedViewMaskCount` is the number of correlation masks.
- `pCorrelatedViewMasks` is a pointer to an array of view masks indicating sets of views that **may** be more efficient to render concurrently.

Parameters defined by this structure with the same name as those in `VkRenderPassCreateInfo` have the identical effect to those parameters; the child structures are variants of those used in `VkRenderPassCreateInfo` which add `sType` and `pNext` parameters, allowing them to be extended.

If the `VkSubpassDescription2::viewMask` member of any element of `pSubpasses` is not zero, *multiview* functionality is considered to be enabled for this render pass.

`correlatedViewMaskCount` and `pCorrelatedViewMasks` have the same effect as `VkRenderPassMultiviewCreateInfo::correlationMaskCount` and `VkRenderPassMultiviewCreateInfo::pCorrelationMasks`, respectively.

Valid Usage

- VUID-VkRenderPassCreateInfo2-None-03049

If any two subpasses operate on attachments with overlapping ranges of the same `VkDeviceMemory` object, and at least one subpass writes to that area of `VkDeviceMemory`, a subpass dependency **must** be included (either directly or via some intermediate subpasses) between them

- VUID-VkRenderPassCreateInfo2-attachment-03050

If the `attachment` member of any element of `pInputAttachments`, `pColorAttachments`, `pResolveAttachments` or `pDepthStencilAttachment`, or the attachment indexed by any element of `pPreserveAttachments` in any element of `pSubpasses` is bound to a range of a `VkDeviceMemory` object that overlaps with any other attachment in any subpass (including the same subpass), the `VkAttachmentDescription2` structures describing them **must** include `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT` in `flags`

- VUID-VkRenderPassCreateInfo2-attachment-03051

If the `attachment` member of any element of `pInputAttachments`, `pColorAttachments`, `pResolveAttachments` or `pDepthStencilAttachment`, or any element of `pPreserveAttachments` in any element of `pSubpasses` is not `VK_ATTACHMENT_UNUSED`, then it **must** be less than `attachmentCount`

- VUID-VkRenderPassCreateInfo2-pSubpasses-06473

If the `pSubpasses` `pNext` chain includes a `VkSubpassDescriptionDepthStencilResolve` structure and the `pDepthStencilResolveAttachment` member is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED`, then `attachment` **must** be less than `attachmentCount`

- VUID-VkRenderPassCreateInfo2-pAttachments-02522

For any member of `pAttachments` with a `loadOp` equal to `VK_ATTACHMENT_LOAD_OP_CLEAR`, the first use of that attachment **must** not specify a `layout` equal to `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`,
`VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, or
`VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`

- VUID-VkRenderPassCreateInfo2-pAttachments-02523

For any member of `pAttachments` with a `stencilLoadOp` equal to `VK_ATTACHMENT_LOAD_OP_CLEAR`, the first use of that attachment **must** not specify a `layout` equal to `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`,
`VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, or
`VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`

- VUID-VkRenderPassCreateInfo2-pDependencies-03054

For any element of `pDependencies`, if the `srcSubpass` is not `VK_SUBPASS_EXTERNAL`, all stage flags included in the `srcStageMask` member of that dependency **must** be `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT` or a pipeline stage supported by the `pipeline` identified by the `pipelineBindPoint` member of the source subpass

- VUID-VkRenderPassCreateInfo2-pDependencies-03055

For any element of `pDependencies`, if the `dstSubpass` is not `VK_SUBPASS_EXTERNAL`, all stage flags included in the `dstStageMask` member of that dependency **must** be `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT` or a pipeline stage supported by the `pipeline`

identified by the `pipelineBindPoint` member of the destination subpass

- VUID-VkRenderPassCreateInfo2-pCorrelatedViewMasks-03056
The set of bits included in any element of `pCorrelatedViewMasks` **must** not overlap with the set of bits included in any other element of `pCorrelatedViewMasks`
- VUID-VkRenderPassCreateInfo2-viewMask-03057
If the `VkSubpassDescription2::viewMask` member of all elements of `pSubpasses` is 0, `correlatedViewMaskCount` **must** be 0
- VUID-VkRenderPassCreateInfo2-viewMask-03058
The `VkSubpassDescription2::viewMask` member of all elements of `pSubpasses` **must** either all be 0, or all not be 0
- VUID-VkRenderPassCreateInfo2-viewMask-03059
If the `VkSubpassDescription2::viewMask` member of all elements of `pSubpasses` is 0, the `dependencyFlags` member of any element of `pDependencies` **must** not include `VK_DEPENDENCY_VIEW_LOCAL_BIT`
- VUID-VkRenderPassCreateInfo2-pDependencies-03060
For any element of `pDependencies` where its `srcSubpass` member equals its `dstSubpass` member, if the `viewMask` member of the corresponding element of `pSubpasses` includes more than one bit, its `dependencyFlags` member **must** include `VK_DEPENDENCY_VIEW_LOCAL_BIT`
- VUID-VkRenderPassCreateInfo2-attachment-02525
If the `attachment` member of any element of the `pInputAttachments` member of any element of `pSubpasses` is not `VK_ATTACHMENT_UNUSED`, the `aspectMask` member of that element of `pInputAttachments` **must** only include aspects that are present in images of the format specified by the element of `pAttachments` specified by `attachment`
- VUID-VkRenderPassCreateInfo2-srcSubpass-02526
The `srcSubpass` member of each element of `pDependencies` **must** be less than `subpassCount`
- VUID-VkRenderPassCreateInfo2-dstSubpass-02527
The `dstSubpass` member of each element of `pDependencies` **must** be less than `subpassCount`
- VUID-VkRenderPassCreateInfo2-pAttachments-04585
If any element of `pAttachments` is used as a fragment shading rate attachment in any subpass, it **must** not be used as any other attachment in the render pass
- VUID-VkRenderPassCreateInfo2-pAttachments-09387
If any element of `pAttachments` is used as a fragment shading rate attachment, the `loadOp` for that attachment **must** not be `VK_ATTACHMENT_LOAD_OP_CLEAR`
- VUID-VkRenderPassCreateInfo2-pAttachments-04586
If any element of `pAttachments` is used as a fragment shading rate attachment in any subpass, it **must** have an image format whose `potential format features` contain `VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-VkRenderPassCreateInfo2-subpassCount-05055
`subpassCount` **must** be less than or equal to `maxRenderPassSubpasses`
- VUID-VkRenderPassCreateInfo2-dependencyCount-05056
`dependencyCount` **must** be less than or equal to `maxRenderPassDependencies`

- VUID-VkRenderPassCreateInfo2-attachmentCount-05057
`attachmentCount` **must** be less than or equal to `maxFramebufferAttachments`
- VUID-VkRenderPassCreateInfo2-attachment-06244
If the `attachment` member of the `pDepthStencilAttachment` member of an element of `pSubpasses` is not `VK_ATTACHMENT_UNUSED`, the `layout` member of that same structure is either `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, and the `pNext` chain of that structure does not include a `VkAttachmentReferenceStencilLayout` structure, then the element of `pAttachments` with an index equal to `attachment` **must** not have a `format` that includes both depth and stencil components
- VUID-VkRenderPassCreateInfo2-attachment-06245
If the `attachment` member of the `pDepthStencilAttachment` member of an element of `pSubpasses` is not `VK_ATTACHMENT_UNUSED` and the `layout` member of that same structure is either `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`, then the element of `pAttachments` with an index equal to `attachment` **must** have a `format` that includes only a stencil component
- VUID-VkRenderPassCreateInfo2-attachment-06246
If the `attachment` member of the `pDepthStencilAttachment` member of an element of `pSubpasses` is not `VK_ATTACHMENT_UNUSED` and the `layout` member of that same structure is either `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, then the element of `pAttachments` with an index equal to `attachment` **must** not have a `format` that includes only a stencil component
- VUID-VkRenderPassCreateInfo2-None-10916
If any subpass preserves an attachment, there **must** be a subpass dependency from a prior subpass which uses or preserves that attachment

Valid Usage (Implicit)

- VUID-VkRenderPassCreateInfo2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO_2`
- VUID-VkRenderPassCreateInfo2-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkRenderPassCreateInfo2-flags-zero bitmask
`flags` **must** be `0`
- VUID-VkRenderPassCreateInfo2-pAttachments-parameter
If `attachmentCount` is not `0`, `pAttachments` **must** be a valid pointer to an array of `attachmentCount` valid `VkAttachmentDescription2` structures
- VUID-VkRenderPassCreateInfo2-pSubpasses-parameter
`pSubpasses` **must** be a valid pointer to an array of `subpassCount` valid `VkSubpassDescription2` structures
- VUID-VkRenderPassCreateInfo2-pDependencies-parameter
If `dependencyCount` is not `0`, `pDependencies` **must** be a valid pointer to an array of `dependencyCount` valid `VkSubpassDependency2` structures

- VUID-VkRenderPassCreateInfo2-pCorrelatedViewMasks-parameter
If `correlatedViewMaskCount` is not 0, `pCorrelatedViewMasks` **must** be a valid pointer to an array of `correlatedViewMaskCount` `uint32_t` values
- VUID-VkRenderPassCreateInfo2-subpassCount-arraylength
`subpassCount` **must** be greater than 0

The `VkAttachmentDescription2` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkAttachmentDescription2 {
    VkStructureType      sType;
    const void*          pNext;
    VkAttachmentDescriptionFlags flags;
    VkFormat              format;
    VkSampleCountFlagBits samples;
    VkAttachmentLoadOp    loadOp;
    VkAttachmentStoreOp   storeOp;
    VkAttachmentLoadOp    stencilLoadOp;
    VkAttachmentStoreOp   stencilStoreOp;
    VkImageLayout         initialLayout;
    VkImageLayout         finalLayout;
} VkAttachmentDescription2;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkAttachmentDescriptionFlagBits` specifying additional properties of the attachment.
- `format` is a `VkFormat` value specifying the format of the image that will be used for the attachment.
- `samples` is a `VkSampleCountFlagBits` value specifying the number of samples of the image.
- `loadOp` is a `VkAttachmentLoadOp` value specifying how the contents of color and depth components of the attachment are treated at the beginning of the subpass where it is first used.
- `storeOp` is a `VkAttachmentStoreOp` value specifying how the contents of color and depth components of the attachment are treated at the end of the subpass where it is last used.
- `stencilLoadOp` is a `VkAttachmentLoadOp` value specifying how the contents of stencil components of the attachment are treated at the beginning of the subpass where it is first used.
- `stencilStoreOp` is a `VkAttachmentStoreOp` value specifying how the contents of stencil components of the attachment are treated at the end of the last subpass where it is used.
- `initialLayout` is the layout the attachment image subresource will be in when a render pass instance begins.
- `finalLayout` is the layout the attachment image subresource will be transitioned to when a render pass instance ends.

Parameters defined by this structure with the same name as those in [VkAttachmentDescription](#) have the identical effect to those parameters.

If the `separateDepthStencilLayouts` feature is enabled, and `format` is a depth/stencil format, `initialLayout` and `finalLayout` **can** be set to a layout that only specifies the layout of the depth aspect.

If the `pNext` chain includes a [VkAttachmentDescriptionStencilLayout](#) structure, then the `stencilInitialLayout` and `stencilFinalLayout` members specify the initial and final layouts of the stencil aspect of a depth/stencil format, and `initialLayout` and `finalLayout` only apply to the depth aspect. For depth-only formats, the [VkAttachmentDescriptionStencilLayout](#) structure is ignored. For stencil-only formats, the initial and final layouts of the stencil aspect are taken from the [VkAttachmentDescriptionStencilLayout](#) structure if present, or `initialLayout` and `finalLayout` if not present.

If `format` is a depth/stencil format, and either `initialLayout` or `finalLayout` does not specify a layout for the stencil aspect, then the application **must** specify the initial and final layouts of the stencil aspect by including a [VkAttachmentDescriptionStencilLayout](#) structure in the `pNext` chain.

`loadOp` and `storeOp` are ignored for fragment shading rate attachments. No access to the shading rate attachment is performed in `loadOp` and `storeOp`. Instead, access to `VK_ACCESS_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR` is performed as fragments are rasterized.

Valid Usage

- VUID-VkAttachmentDescription2-format-06699
If `format` includes a color or depth component and `loadOp` is `VK_ATTACHMENT_LOAD_OP_LOAD`, then `initialLayout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED`
- VUID-VkAttachmentDescription2-finalLayout-00843
`finalLayout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`
- VUID-VkAttachmentDescription2-format-03280
If `format` is a color format, `initialLayout` **must** not be `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`
- VUID-VkAttachmentDescription2-format-03281
If `format` is a depth/stencil format, `initialLayout` **must** not be `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`
- VUID-VkAttachmentDescription2-format-03282
If `format` is a color format, `finalLayout` **must** not be `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`
- VUID-VkAttachmentDescription2-format-03283
If `format` is a depth/stencil format, `finalLayout` **must** not be `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`
- VUID-VkAttachmentDescription2-format-06487
If `format` is a color format, `initialLayout` **must** not be

VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL or
VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL

- VUID-VkAttachmentDescription2-format-06488

If `format` is a color format, `finalLayout` **must** not be
VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL or
VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL

- VUID-VkAttachmentDescription2-separateDepthStencilLayouts-03284

If the `separateDepthStencilLayouts` feature is not enabled, `initialLayout` **must** not be
VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL,
VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL,

- VUID-VkAttachmentDescription2-separateDepthStencilLayouts-03285

If the `separateDepthStencilLayouts` feature is not enabled, `finalLayout` **must** not be
VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL,
VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL,

- VUID-VkAttachmentDescription2-format-03286

If `format` is a color format, `initialLayout` **must** not be
VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL,
VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription2-format-03287

If `format` is a color format, `finalLayout` **must** not be
VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL,
VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription2-format-06906

If `format` is a depth/stencil format which includes both depth and stencil components,
`initialLayout` **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription2-format-06907

If `format` is a depth/stencil format which includes both depth and stencil components,
`finalLayout` **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription2-format-03290

If `format` is a depth/stencil format which includes only the depth component,
`initialLayout` **must** not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription2-format-03291

If `format` is a depth/stencil format which includes only the depth component, `finalLayout`
must not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL or
VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription2-synchronization2-06908

If the `synchronization2` feature is not enabled, `initialLayout` **must** not be

VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR

- VUID-VkAttachmentDescription2-synchronization2-06909

If the `synchronization2` feature is not enabled, `finalLayout` **must** not be VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR or VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR

- VUID-VkAttachmentDescription2-samples-08745

`samples` **must** be a valid `VkSampleCountFlagBits` value that is set in `imageCreateSampleCounts` (as defined in [Image Creation Limits](#)) for the given `format`

- VUID-VkAttachmentDescription2-pNext-06704

If the `pNext` chain does not include a `VkAttachmentDescriptionStencilLayout` structure, `format` includes a stencil component, and `stencilLoadOp` is VK_ATTACHMENT_LOAD_OP_LOAD, then `initialLayout` **must** not be VK_IMAGE_LAYOUT_UNDEFINED

- VUID-VkAttachmentDescription2-pNext-06705

If the `pNext` chain includes a `VkAttachmentDescriptionStencilLayout` structure, `format` includes a stencil component, and `stencilLoadOp` is VK_ATTACHMENT_LOAD_OP_LOAD, then `VkAttachmentDescriptionStencilLayout::stencilInitialLayout` **must** not be VK_IMAGE_LAYOUT_UNDEFINED

- VUID-VkAttachmentDescription2-format-06249

If `format` is a depth/stencil format which includes both depth and stencil components, and `initialLayout` is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, the `pNext` chain **must** include a `VkAttachmentDescriptionStencilLayout` structure

- VUID-VkAttachmentDescription2-format-06250

If `format` is a depth/stencil format which includes both depth and stencil components, and `finalLayout` is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, the `pNext` chain **must** include a `VkAttachmentDescriptionStencilLayout` structure

- VUID-VkAttachmentDescription2-format-06247

If the `pNext` chain does not include a `VkAttachmentDescriptionStencilLayout` structure and `format` only includes a stencil component, `initialLayout` **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription2-format-06248

If the `pNext` chain does not include a `VkAttachmentDescriptionStencilLayout` structure and `format` only includes a stencil component, `finalLayout` **must** not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL

- VUID-VkAttachmentDescription2-format-09332

`format` **must** not be VK_FORMAT_UNDEFINED

Valid Usage (Implicit)

- VUID-VkAttachmentDescription2-sType-sType

`sType` **must** be VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2

- VUID-VkAttachmentDescription2-pNext-pNext

`pNext` **must** be NULL or a pointer to a valid instance of

VkAttachmentDescriptionStencilLayout

- VUID-VkAttachmentDescription2-sType-unique
The **sType** value of each structure in the **pNext** chain **must** be unique
- VUID-VkAttachmentDescription2-flags-parameter
flags **must** be a valid combination of **VkAttachmentDescriptionFlagBits** values
- VUID-VkAttachmentDescription2-format-parameter
format **must** be a valid **VkFormat** value
- VUID-VkAttachmentDescription2-samples-parameter
samples **must** be a valid **VkSampleCountFlagBits** value
- VUID-VkAttachmentDescription2-loadOp-parameter
loadOp **must** be a valid **VkAttachmentLoadOp** value
- VUID-VkAttachmentDescription2-storeOp-parameter
storeOp **must** be a valid **VkAttachmentStoreOp** value
- VUID-VkAttachmentDescription2-stencilLoadOp-parameter
stencilLoadOp **must** be a valid **VkAttachmentLoadOp** value
- VUID-VkAttachmentDescription2-stencilStoreOp-parameter
stencilStoreOp **must** be a valid **VkAttachmentStoreOp** value
- VUID-VkAttachmentDescription2-initialLayout-parameter
initialLayout **must** be a valid **VkImageLayout** value
- VUID-VkAttachmentDescription2-finalLayout-parameter
finalLayout **must** be a valid **VkImageLayout** value

The **VkAttachmentDescriptionStencilLayout** structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkAttachmentDescriptionStencilLayout {
    VkStructureType    sType;
    void*              pNext;
    VkImageLayout       stencilInitialLayout;
    VkImageLayout       stencilFinalLayout;
} VkAttachmentDescriptionStencilLayout;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **stencilInitialLayout** is the layout the stencil aspect of the attachment image subresource will be in when a render pass instance begins.
- **stencilFinalLayout** is the layout the stencil aspect of the attachment image subresource will be transitioned to when a render pass instance ends.

Valid Usage

- VUID-VkAttachmentDescriptionStencilLayout-stencilInitialLayout-03308
stencilInitialLayout **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkAttachmentDescriptionStencilLayout-stencilFinalLayout-03309
stencilFinalLayout **must** not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- VUID-VkAttachmentDescriptionStencilLayout-stencilFinalLayout-03310
stencilFinalLayout **must** not be VK_IMAGE_LAYOUT_UNDEFINED or VK_IMAGE_LAYOUT_PREINITIALIZED

Valid Usage (Implicit)

- VUID-VkAttachmentDescriptionStencilLayout-sType-sType
sType **must** be VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_STENCIL_LAYOUT
- VUID-VkAttachmentDescriptionStencilLayout-stencilInitialLayout-parameter
stencilInitialLayout **must** be a valid [VkImageLayout](#) value
- VUID-VkAttachmentDescriptionStencilLayout-stencilFinalLayout-parameter
stencilFinalLayout **must** be a valid [VkImageLayout](#) value

The [VkSubpassDescription2](#) structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkSubpassDescription2 {
    VkStructureType           sType;
    const void*               pNext;
    VkSubpassDescriptionFlags flags;
    VkPipelineBindPoint       pipelineBindPoint;
    uint32_t                  viewMask;
    uint32_t                  inputAttachmentCount;
    const VkAttachmentReference2* pInputAttachments;
    uint32_t                  colorAttachmentCount;
    const VkAttachmentReference2* pColorAttachments;
    const VkAttachmentReference2* pResolveAttachments;
    const VkAttachmentReference2* pDepthStencilAttachment;
    uint32_t                  preserveAttachmentCount;
    const uint32_t*           pPreserveAttachments;
} VkSubpassDescription2;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **flags** is a bitmask of [VkSubpassDescriptionFlagBits](#) specifying usage of the subpass.
- **pipelineBindPoint** is a [VkPipelineBindPoint](#) value specifying the pipeline type supported for this subpass.
- **viewMask** is a bitfield of view indices describing which views rendering is broadcast to in this subpass, when multiview is enabled.
- **inputAttachmentCount** is the number of input attachments.
- **pInputAttachments** is a pointer to an array of [VkAttachmentReference2](#) structures defining the input attachments for this subpass and their layouts.
- **colorAttachmentCount** is the number of color attachments.
- **pColorAttachments** is a pointer to an array of **colorAttachmentCount** [VkAttachmentReference2](#) structures defining the color attachments for this subpass and their layouts.
- **pResolveAttachments** is **NULL** or a pointer to an array of **colorAttachmentCount** [VkAttachmentReference2](#) structures defining the resolve attachments for this subpass and their layouts.
- **pDepthStencilAttachment** is a pointer to a [VkAttachmentReference2](#) structure specifying the depth/stencil attachment for this subpass and its layout.
- **preserveAttachmentCount** is the number of preserved attachments.
- **pPreserveAttachments** is a pointer to an array of **preserveAttachmentCount** render pass attachment indices identifying attachments that are not used by this subpass, but whose contents **must** be preserved throughout the subpass.

Parameters defined by this structure with the same name as those in [VkSubpassDescription](#) have the identical effect to those parameters.

`viewMask` has the same effect for the described subpass as `VkRenderPassMultiviewCreateInfo::pViewMasks` has on each corresponding subpass.

If a `VkFragmentShadingRateAttachmentInfoKHR` structure is included in the `pNext` chain, `pFragmentShadingRateAttachment` is not `NULL`, and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, the identified attachment defines a fragment shading rate attachment for that subpass.

Valid Usage

- VUID-VkSubpassDescription2-attachment-06912
If the `attachment` member of an element of `pInputAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL`
- VUID-VkSubpassDescription2-attachment-06913
If the `attachment` member of an element of `pColorAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`
- VUID-VkSubpassDescription2-attachment-06914
If the `attachment` member of an element of `pResolveAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`
- VUID-VkSubpassDescription2-attachment-06915
If the `attachment` member of `pDepthStencilAttachment` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`
- VUID-VkSubpassDescription2-attachment-06916
If the `attachment` member of an element of `pColorAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`
- VUID-VkSubpassDescription2-attachment-06917
If the `attachment` member of an element of `pResolveAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`
- VUID-VkSubpassDescription2-attachment-06918
If the `attachment` member of an element of `pInputAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`
- VUID-VkSubpassDescription2-attachment-06919
If the `attachment` member of an element of `pColorAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`

- VUID-VkSubpassDescription2-attachment-06920
If the `attachment` member of an element of `pResolveAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- VUID-VkSubpassDescription2-attachment-06921
If the `attachment` member of an element of `pInputAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR`
- VUID-VkSubpassDescription2-attachment-06922
If the `attachment` member of an element of `pColorAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`
- VUID-VkSubpassDescription2-attachment-06923
If the `attachment` member of an element of `pResolveAttachments` is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** not be `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`
- VUID-VkSubpassDescription2-attachment-06251
If the `attachment` member of `pDepthStencilAttachment` is not `VK_ATTACHMENT_UNUSED` and its `pNext` chain includes a `VkAttachmentReferenceStencilLayout` structure, the `layout` member of `pDepthStencilAttachment` **must** not be `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- VUID-VkSubpassDescription2-pipelineBindPoint-04953
`pipelineBindPoint` **must** be `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-VkSubpassDescription2-colorAttachmentCount-03063
`colorAttachmentCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxColorAttachments`
- VUID-VkSubpassDescription2-loadOp-03064
If the first use of an attachment in this render pass is as an input attachment, and the attachment is not also used as a color or depth/stencil attachment in the same subpass, then `loadOp` **must** not be `VK_ATTACHMENT_LOAD_OP_CLEAR`
- VUID-VkSubpassDescription2-pResolveAttachments-03065
If `pResolveAttachments` is not `NULL`, for each resolve attachment that does not have the value `VK_ATTACHMENT_UNUSED`, the corresponding color attachment **must** not have the value `VK_ATTACHMENT_UNUSED`
- VUID-VkSubpassDescription2-pResolveAttachments-03066
If `pResolveAttachments` is not `NULL`, for each resolve attachment that is not `VK_ATTACHMENT_UNUSED`, the corresponding color attachment **must** not have a sample count of `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkSubpassDescription2-pResolveAttachments-03068
Each element of `pResolveAttachments` **must** have the same `VkFormat` as its corresponding color attachment
- VUID-VkSubpassDescription2-pResolveAttachments-03067

If `pResolveAttachments` is not `NULL`, each resolve attachment that is not `VK_ATTACHMENT_UNUSED` **must** have a sample count of `VK_SAMPLE_COUNT_1_BIT`

- VUID-VkSubpassDescription2-pInputAttachments-02897

All attachments in `pInputAttachments` that are not `VK_ATTACHMENT_UNUSED`

must have image formats whose `potential format features` contain at least `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` or `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-VkSubpassDescription2-pColorAttachments-02898

All attachments in `pColorAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have image formats whose `potential format features` contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`

- VUID-VkSubpassDescription2-pResolveAttachments-02899

All attachments in `pResolveAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have image formats whose `potential format features` contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`

- VUID-VkSubpassDescription2-pDepthStencilAttachment-02900

If `pDepthStencilAttachment` is not `NULL` and the attachment is not `VK_ATTACHMENT_UNUSED` then it **must** have an image format whose `potential format features` contain `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-VkSubpassDescription2-multisampledRenderToSingleSampled-06872

All attachments in `pDepthStencilAttachment` or `pColorAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have the same sample count

- VUID-VkSubpassDescription2-attachment-03073

Each element of `pPreserveAttachments` **must** not be `VK_ATTACHMENT_UNUSED`

- VUID-VkSubpassDescription2-pPreserveAttachments-03074

Each element of `pPreserveAttachments` **must** not also be an element of any other member of the subpass description

- VUID-VkSubpassDescription2-layout-02528

If any attachment is used by more than one `VkAttachmentReference2` member, then each use **must** use the same `layout`

- VUID-VkSubpassDescription2-attachment-02799

If the `attachment` member of any element of `pInputAttachments` is not `VK_ATTACHMENT_UNUSED`, then the `aspectMask` member **must** be a valid combination of `VkImageAspectFlagBits`

- VUID-VkSubpassDescription2-attachment-02800

If the `attachment` member of any element of `pInputAttachments` is not `VK_ATTACHMENT_UNUSED`, then the `aspectMask` member **must** not be `0`

- VUID-VkSubpassDescription2-attachment-02801

If the `attachment` member of any element of `pInputAttachments` is not `VK_ATTACHMENT_UNUSED`, then the `aspectMask` member **must** not include `VK_IMAGE_ASPECT_METADATA_BIT`

- VUID-VkSubpassDescription2-attachment-04563

If the `attachment` member of any element of `pInputAttachments` is not `VK_ATTACHMENT_UNUSED`, then the `aspectMask` member **must** not include `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT` for any index `i`

- VUID-VkSubpassDescription2-pDepthStencilAttachment-04440
An attachment **must** not be used in both `pDepthStencilAttachment` and `pColorAttachments`
- VUID-VkSubpassDescription2-inputAttachmentCount-05058
`inputAttachmentCount` **must** be less than or equal to `maxSubpassInputAttachments`
- VUID-VkSubpassDescription2-preserveAttachmentCount-05059
`preserveAttachmentCount` **must** be less than or equal to `maxSubpassPreserveAttachments`
- VUID-VkSubpassDescription2-multiview-06558
If the `multiview` feature is not enabled, `viewMask` **must** be 0
- VUID-VkSubpassDescription2-viewMask-06706
The index of the most significant bit in `viewMask` **must** be less than `maxMultiviewViewCount`

Valid Usage (Implicit)

- VUID-VkSubpassDescription2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_2`
- VUID-VkSubpassDescription2-pNext-pNext
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkFragmentShadingRateAttachmentInfoKHR` or `VkSubpassDescriptionDepthStencilResolve`
- VUID-VkSubpassDescription2-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkSubpassDescription2-flags-zero bitmask
`flags` **must** be 0
- VUID-VkSubpassDescription2-pipelineBindPoint-parameter
`pipelineBindPoint` **must** be a valid `VkPipelineBindPoint` value
- VUID-VkSubpassDescription2-pInputAttachments-parameter
If `inputAttachmentCount` is not 0, `pInputAttachments` **must** be a valid pointer to an array of `inputAttachmentCount` valid `VkAttachmentReference2` structures
- VUID-VkSubpassDescription2-pColorAttachments-parameter
If `colorAttachmentCount` is not 0, `pColorAttachments` **must** be a valid pointer to an array of `colorAttachmentCount` valid `VkAttachmentReference2` structures
- VUID-VkSubpassDescription2-pResolveAttachments-parameter
If `colorAttachmentCount` is not 0, and `pResolveAttachments` is not `NULL`, `pResolveAttachments` **must** be a valid pointer to an array of `colorAttachmentCount` valid `VkAttachmentReference2` structures
- VUID-VkSubpassDescription2-pDepthStencilAttachment-parameter
If `pDepthStencilAttachment` is not `NULL`, `pDepthStencilAttachment` **must** be a valid pointer to a valid `VkAttachmentReference2` structure
- VUID-VkSubpassDescription2-pPreserveAttachments-parameter
If `preserveAttachmentCount` is not 0, `pPreserveAttachments` **must** be a valid pointer to an array of `preserveAttachmentCount` `uint32_t` values

The `VkSubpassDescriptionDepthStencilResolve` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkSubpassDescriptionDepthStencilResolve {
    VkStructureType           sType;
    const void*               pNext;
    VkResolveModeFlagBits     depthResolveMode;
    VkResolveModeFlagBits     stencilResolveMode;
    const VkAttachmentReference2* pDepthStencilResolveAttachment;
} VkSubpassDescriptionDepthStencilResolve;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `depthResolveMode` is a `VkResolveModeFlagBits` value describing the depth resolve mode.
- `stencilResolveMode` is a `VkResolveModeFlagBits` value describing the stencil resolve mode.
- `pDepthStencilResolveAttachment` is `NULL` or a pointer to a `VkAttachmentReference2` structure defining the depth/stencil resolve attachment for this subpass and its layout.

If the `pNext` chain of `VkSubpassDescription2` includes a `VkSubpassDescriptionDepthStencilResolve` structure, then that structure describes [multisample resolve operations](#) for the depth/stencil attachment in a subpass. If this structure is not included in the `pNext` chain of `VkSubpassDescription2`, or if it is and either `pDepthStencilResolveAttachment` is `NULL` or its attachment index is `VK_ATTACHMENT_UNUSED`, it indicates that no depth/stencil resolve attachment will be used in the subpass.

Valid Usage

- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03177
If `pDepthStencilResolveAttachment` is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED`, `pDepthStencilAttachment` **must** not be `NULL` or have the value `VK_ATTACHMENT_UNUSED`
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03179
If `pDepthStencilResolveAttachment` is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED`, `pDepthStencilAttachment` **must** not have a sample count of `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03180
If `pDepthStencilResolveAttachment` is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED`, `pDepthStencilResolveAttachment` **must** have a sample count of `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-02651
If `pDepthStencilResolveAttachment` is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED` then it **must** have an image format whose [potential format features](#) contain `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT`
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03181

If `pDepthStencilResolveAttachment` is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED` and `VkFormat` of `pDepthStencilResolveAttachment` has a depth component, then the `VkFormat` of `pDepthStencilAttachment` **must** have a depth component with the same number of bits and `numeric format`

- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03182
If `pDepthStencilResolveAttachment` is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED`, and `VkFormat` of `pDepthStencilResolveAttachment` has a stencil component, then the `VkFormat` of `pDepthStencilAttachment` **must** have a stencil component with the same number of bits and `numeric format`
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03178
If `pDepthStencilResolveAttachment` is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED`, `depthResolveMode` and `stencilResolveMode` **must** not both be `VK_RESOLVE_MODE_NONE`
- VUID-VkSubpassDescriptionDepthStencilResolve-depthResolveMode-03183
If `pDepthStencilResolveAttachment` is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED` and the `VkFormat` of `pDepthStencilResolveAttachment` has a depth component, then the value of `depthResolveMode` **must** be one of the bits set in `VkPhysicalDeviceDepthStencilResolveProperties::supportedDepthResolveModes` or `VK_RESOLVE_MODE_NONE`
- VUID-VkSubpassDescriptionDepthStencilResolve-stencilResolveMode-03184
If `pDepthStencilResolveAttachment` is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED` and the `VkFormat` of `pDepthStencilResolveAttachment` has a stencil component, then the value of `stencilResolveMode` **must** be one of the bits set in `VkPhysicalDeviceDepthStencilResolveProperties::supportedStencilResolveModes` or `VK_RESOLVE_MODE_NONE`
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03185
If `pDepthStencilResolveAttachment` is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED`, the `VkFormat` of `pDepthStencilResolveAttachment` has both depth and stencil components, `VkPhysicalDeviceDepthStencilResolveProperties::independentResolve` is `VK_FALSE`, and `VkPhysicalDeviceDepthStencilResolveProperties::independentResolveNone` is `VK_FALSE`, then the values of `depthResolveMode` and `stencilResolveMode` **must** be identical
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-03186
If `pDepthStencilResolveAttachment` is not `NULL` and does not have the value `VK_ATTACHMENT_UNUSED`, the `VkFormat` of `pDepthStencilResolveAttachment` has both depth and stencil components, `VkPhysicalDeviceDepthStencilResolveProperties::independentResolve` is `VK_FALSE` and `VkPhysicalDeviceDepthStencilResolveProperties::independentResolveNone` is `VK_TRUE`, then the values of `depthResolveMode` and `stencilResolveMode` **must** be identical or one of them **must** be `VK_RESOLVE_MODE_NONE`

Valid Usage (Implicit)

- VUID-VkSubpassDescriptionDepthStencilResolve-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_DEPTH_STENCIL_RESOLVE`
- VUID-VkSubpassDescriptionDepthStencilResolve-pDepthStencilResolveAttachment-

parameter

If `pDepthStencilResolveAttachment` is not `NULL`, `pDepthStencilResolveAttachment` **must** be a valid pointer to a valid `VkAttachmentReference2` structure

The `VkFragmentShadingRateAttachmentInfoKHR` structure is defined as:

```
// Provided by VK_KHR_fragment_shading_rate
typedef struct VkFragmentShadingRateAttachmentInfoKHR {
    VkStructureType           sType;
    const void*               pNext;
    const VkAttachmentReference2* pFragmentShadingRateAttachment;
    VkExtent2D                shadingRateAttachmentTexelSize;
} VkFragmentShadingRateAttachmentInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pFragmentShadingRateAttachment` is `NULL` or a pointer to a `VkAttachmentReference2` structure defining the fragment shading rate attachment for this subpass.
- `shadingRateAttachmentTexelSize` specifies the size of the portion of the framebuffer corresponding to each texel in `pFragmentShadingRateAttachment`.

If no shading rate attachment is specified, or if this structure is not specified, the implementation behaves as if a valid shading rate attachment was specified with all texels specifying a single pixel per fragment.

Valid Usage

- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04524
If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, its `layout` member **must** be equal to `VK_IMAGE_LAYOUT_GENERAL` or `VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR`
- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04525
If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, `shadingRateAttachmentTexelSize.width` **must** be a power of two value
- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04526
If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, `shadingRateAttachmentTexelSize.width` **must** be less than or equal to `maxFragmentShadingRateAttachmentTexelSize.width`
- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04527

If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, `shadingRateAttachmentTexelSize.width` **must** be greater than or equal to `minFragmentShadingRateAttachmentTexelSize.width`

- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04528

If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, `shadingRateAttachmentTexelSize.height` **must** be a power of two value

- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04529

If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, `shadingRateAttachmentTexelSize.height` **must** be less than or equal to `maxFragmentShadingRateAttachmentTexelSize.height`

- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04530

If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, `shadingRateAttachmentTexelSize.height` **must** be greater than or equal to `minFragmentShadingRateAttachmentTexelSize.height`

- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04531

If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, the quotient of `shadingRateAttachmentTexelSize.width` and `shadingRateAttachmentTexelSize.height` **must** be less than or equal to `maxFragmentShadingRateAttachmentTexelSizeAspectRatio`

- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-04532

If `pFragmentShadingRateAttachment` is not `NULL` and its `attachment` member is not `VK_ATTACHMENT_UNUSED`, the quotient of `shadingRateAttachmentTexelSize.height` and `shadingRateAttachmentTexelSize.width` **must** be less than or equal to `maxFragmentShadingRateAttachmentTexelSizeAspectRatio`

Valid Usage (Implicit)

- VUID-VkFragmentShadingRateAttachmentInfoKHR-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_FRAGMENT_SHADING_RATE_ATTACHMENT_INFO_KHR`

- VUID-VkFragmentShadingRateAttachmentInfoKHR-pFragmentShadingRateAttachment-parameter

If `pFragmentShadingRateAttachment` is not `NULL`, `pFragmentShadingRateAttachment` **must** be a valid pointer to a valid [VkAttachmentReference2](#) structure

The `VkAttachmentReference2` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkAttachmentReference2 {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           attachment;
    VkImageLayout       layout;
    VkImageAspectFlags aspectMask;
} VkAttachmentReference2;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **attachment** is either an integer value identifying an attachment at the corresponding index in **VkRenderPassCreateInfo2::pAttachments**, or **VK_ATTACHMENT_UNUSED** to signify that this attachment is not used.
- **layout** is a **VkImageLayout** value specifying the layout the attachment uses during the subpass.
- **aspectMask** is a mask of which aspect(s) **can** be accessed within the specified subpass as an input attachment.

Parameters defined by this structure with the same name as those in **VkAttachmentReference** have the identical effect to those parameters.

aspectMask is ignored when this structure is used to describe anything other than an input attachment reference.

If the **separateDepthStencilLayouts** feature is enabled, and **attachment** has a depth/stencil format, **layout** **can** be set to a layout that only specifies the layout of the depth aspect.

If **layout** only specifies the layout of the depth aspect of the attachment, the layout of the stencil aspect is specified by the **stencilLayout** member of a **VkAttachmentReferenceStencilLayout** structure included in the **pNext** chain. Otherwise, **layout** describes the layout for all relevant image aspects.

Valid Usage

- VUID-VkAttachmentReference2-layout-03077
If **attachment** is not **VK_ATTACHMENT_UNUSED**, **layout** **must** not be **VK_IMAGE_LAYOUT_UNDEFINED**, **VK_IMAGE_LAYOUT_PREINITIALIZED**, or **VK_IMAGE_LAYOUT_PRESENT_SRC_KHR**
- VUID-VkAttachmentReference2-separateDepthStencilLayouts-03313
If the **separateDepthStencilLayouts** feature is not enabled, and **attachment** is not **VK_ATTACHMENT_UNUSED**, **layout** **must** not be **VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL**, **VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL**, **VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL**, or **VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL**,
- VUID-VkAttachmentReference2-synchronization2-06910
If the **synchronization2** feature is not enabled, **layout** **must** not be **VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR** or **VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR**

Valid Usage (Implicit)

- VUID-VkAttachmentReference2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2`
- VUID-VkAttachmentReference2-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkAttachmentReferenceStencilLayout`
- VUID-VkAttachmentReference2-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkAttachmentReference2-layout-parameter
`layout` **must** be a valid `VkImageLayout` value

The `VkAttachmentReferenceStencilLayout` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkAttachmentReferenceStencilLayout {
    VkStructureType    sType;
    void*              pNext;
    VkImageLayout       stencilLayout;
} VkAttachmentReferenceStencilLayout;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stencilLayout` is a `VkImageLayout` value specifying the layout the stencil aspect of the attachment uses during the subpass.

Valid Usage

- VUID-VkAttachmentReferenceStencilLayout-stencilLayout-03318
`stencilLayout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED`, `VK_IMAGE_LAYOUT_PREINITIALIZED`, `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_PRESENT_SRC_KHR`

Valid Usage (Implicit)

- VUID-VkAttachmentReferenceStencilLayout-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_STENCIL_LAYOUT`

- VUID-VkAttachmentReferenceStencilLayout-stencilLayout-parameter **stencilLayout** must be a valid [VkImageLayout](#) value

The [VkSubpassDependency2](#) structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkSubpassDependency2 {
    VkStructureType      sType;
    const void*          pNext;
    uint32_t             srcSubpass;
    uint32_t             dstSubpass;
    VkPipelineStageFlags srcStageMask;
    VkPipelineStageFlags dstStageMask;
    VkAccessFlags        srcAccessMask;
    VkAccessFlags        dstAccessMask;
    VkDependencyFlags    dependencyFlags;
    int32_t              viewOffset;
} VkSubpassDependency2;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is `NULL` or a pointer to a structure extending this structure.
- **srcSubpass** is the subpass index of the first subpass in the dependency, or `VK_SUBPASS_EXTERNAL`.
- **dstSubpass** is the subpass index of the second subpass in the dependency, or `VK_SUBPASS_EXTERNAL`.
- **srcStageMask** is a bitmask of [VkPipelineStageFlagBits](#) specifying the [source stage mask](#). If set to `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT`, it is equivalent to setting it to `VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT`.
- **dstStageMask** is a bitmask of [VkPipelineStageFlagBits](#) specifying the [destination stage mask](#). If set to `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT`, it is equivalent to setting it to `VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT`.
- **srcAccessMask** is a bitmask of [VkAccessFlagBits](#) specifying a [source access mask](#).
- **dstAccessMask** is a bitmask of [VkAccessFlagBits](#) specifying a [destination access mask](#).
- **dependencyFlags** is a bitmask of [VkDependencyFlagBits](#).
- **viewOffset** controls which views in the source subpass the views in the destination subpass depend on.

Parameters defined by this structure with the same name as those in [VkSubpassDependency](#) have the identical effect to those parameters.

viewOffset has the same effect for the described subpass dependency as [VkRenderPassMultiviewCreateInfo::pViewOffsets](#) has on each corresponding subpass dependency.

If a [VkMemoryBarrier2](#) is included in the **pNext** chain, **srcStageMask**, **dstStageMask**, **srcAccessMask**, and **dstAccessMask** parameters are ignored. The synchronization and access scopes instead are

defined by the parameters of [VkMemoryBarrier2](#). If either `srcStageMask` or `dstStageMask` are set to `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT`, it is equivalent to setting `VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT`.

Valid Usage

- VUID-VkSubpassDependency2-srcStageMask-04090
If the `geometryShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-VkSubpassDependency2-srcStageMask-04091
If the `tessellationShader` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-VkSubpassDependency2-srcStageMask-07319
If the `attachmentFragmentShadingRate` feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-VkSubpassDependency2-srcStageMask-03937
If the `synchronization2` feature is not enabled, `srcStageMask` **must** not be `0`
- VUID-VkSubpassDependency2-dstStageMask-04090
If the `geometryShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-VkSubpassDependency2-dstStageMask-04091
If the `tessellationShader` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-VkSubpassDependency2-dstStageMask-07319
If the `attachmentFragmentShadingRate` feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-VkSubpassDependency2-dstStageMask-03937
If the `synchronization2` feature is not enabled, `dstStageMask` **must** not be `0`
- VUID-VkSubpassDependency2-srcSubpass-03084
`srcSubpass` **must** be less than or equal to `dstSubpass`, unless one of them is `VK_SUBPASS_EXTERNAL`, to avoid cyclic dependencies and ensure a valid execution order
- VUID-VkSubpassDependency2-srcSubpass-03085
`srcSubpass` and `dstSubpass` **must** not both be equal to `VK_SUBPASS_EXTERNAL`
- VUID-VkSubpassDependency2-srcSubpass-06810
If `srcSubpass` is equal to `dstSubpass` and `srcStageMask` includes a [framebuffer-space stage](#), `dstStageMask` **must** only contain [framebuffer-space stages](#)
- VUID-VkSubpassDependency2-srcAccessMask-03088
Any access flag included in `srcAccessMask` **must** be supported by one of the pipeline stages in `srcStageMask`, as specified in the [table of supported access types](#)
- VUID-VkSubpassDependency2-dstAccessMask-03089
Any access flag included in `dstAccessMask` **must** be supported by one of the pipeline stages

in `dstStageMask`, as specified in the [table of supported access types](#)

- VUID-VkSubpassDependency2-dependencyFlags-03090
If `dependencyFlags` includes `VK_DEPENDENCY_VIEW_LOCAL_BIT`, `srcSubpass` **must** not be equal to `VK_SUBPASS_EXTERNAL`
- VUID-VkSubpassDependency2-dependencyFlags-03091
If `dependencyFlags` includes `VK_DEPENDENCY_VIEW_LOCAL_BIT`, `dstSubpass` **must** not be equal to `VK_SUBPASS_EXTERNAL`
- VUID-VkSubpassDependency2-srcSubpass-02245
If `srcSubpass` equals `dstSubpass`, and `srcStageMask` and `dstStageMask` both include a [framebuffer-space stage](#), then `dependencyFlags` **must** include `VK_DEPENDENCY_BY_REGION_BIT`
- VUID-VkSubpassDependency2-viewOffset-02530
If `viewOffset` is not equal to 0, `srcSubpass` **must** not be equal to `dstSubpass`
- VUID-VkSubpassDependency2-dependencyFlags-03092
If `dependencyFlags` does not include `VK_DEPENDENCY_VIEW_LOCAL_BIT`, `viewOffset` **must** be 0

Valid Usage (Implicit)

- VUID-VkSubpassDependency2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SUBPASS_DEPENDENCY_2`
- VUID-VkSubpassDependency2-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of [VkMemoryBarrier2](#)
- VUID-VkSubpassDependency2-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkSubpassDependency2-srcStageMask-parameter
`srcStageMask` **must** be a valid combination of [VkPipelineStageFlagBits](#) values
- VUID-VkSubpassDependency2-dstStageMask-parameter
`dstStageMask` **must** be a valid combination of [VkPipelineStageFlagBits](#) values
- VUID-VkSubpassDependency2-srcAccessMask-parameter
`srcAccessMask` **must** be a valid combination of [VkAccessFlagBits](#) values
- VUID-VkSubpassDependency2-dstAccessMask-parameter
`dstAccessMask` **must** be a valid combination of [VkAccessFlagBits](#) values
- VUID-VkSubpassDependency2-dependencyFlags-parameter
`dependencyFlags` **must** be a valid combination of [VkDependencyFlagBits](#) values

To destroy a render pass, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyRenderPass(
    VkDevice                device,
    VkRenderPass            renderPass,
    const VkAllocationCallbacks* pAllocator);
```

- **device** is the logical device that destroys the render pass.
- **renderPass** is the handle of the render pass to destroy.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage

- VUID-vkDestroyRenderPass-renderPass-00873
All submitted commands that refer to **renderPass** **must** have completed execution

Valid Usage (Implicit)

- VUID-vkDestroyRenderPass-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkDestroyRenderPass-renderPass-parameter
If **renderPass** is not [VK_NULL_HANDLE](#), **renderPass** **must** be a valid [VkRenderPass](#) handle
- VUID-vkDestroyRenderPass-pAllocator-null
pAllocator **must** be [NULL](#)
- VUID-vkDestroyRenderPass-renderPass-parent
If **renderPass** is a valid handle, it **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to **renderPass** **must** be externally synchronized

8.2. Render Pass Compatibility

Framebuffers and graphics pipelines are created based on a specific render pass object. They **must** only be used with that render pass object, or one compatible with it.

Two attachment references are compatible if they have matching format and sample count, or are both [VK_ATTACHMENT_UNUSED](#).

Two arrays of attachment references are compatible if all corresponding pairs of attachments are compatible. If the arrays are of different lengths, attachment references not present in the smaller array are treated as [VK_ATTACHMENT_UNUSED](#). If the pointer that would contain the reference is [NULL](#), the attachment reference is treated as [VK_ATTACHMENT_UNUSED](#).

Two render passes are compatible if their corresponding color, input, resolve, and depth/stencil attachment references are compatible and if they are otherwise identical except for:

- Initial and final image layout in attachment descriptions
- Load and store operations in attachment descriptions

- Image layout in attachment references

As an additional special case, if two render passes have a single subpass, the resolve attachment reference compatibility requirements are ignored.

A framebuffer is compatible with a render pass if it was created using the same render pass or a compatible render pass.

8.3. Framebuffers

Render passes operate in conjunction with *framebuffers*. Framebuffers represent a collection of specific memory attachments that a render pass instance uses.

Framebuffers are represented by `VkFramebuffer` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkFramebuffer)
```

To create a framebuffer, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateFramebuffer(
    VkDevice                                device,
    const VkFramebufferCreateInfo*          pCreateInfo,
    const VkAllocationCallbacks*           pAllocator,
    VkFramebuffer*                          pFramebuffer);
```

- `device` is the logical device that creates the framebuffer.
- `pCreateInfo` is a pointer to a `VkFramebufferCreateInfo` structure describing additional information about framebuffer creation.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pFramebuffer` is a pointer to a `VkFramebuffer` handle in which the resulting framebuffer object is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateFramebuffer` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateFramebuffer-device-10002
`device` **must** support at least one queue family with the `VK_QUEUE_GRAPHICS_BIT` capability
- VUID-vkCreateFramebuffer-pCreateInfo-02777
If `pCreateInfo->flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, and `attachmentCount` is not 0, each element of `pCreateInfo->pAttachments` **must** have been created on `device`

- VUID-vkCreateFramebuffer-device-05068

The number of framebuffers currently allocated from **device** plus 1 **must** be less than or equal to the total number of framebuffers requested via `VkDeviceObjectReservationCreateInfo::framebufferRequestCount` specified when **device** was created

Valid Usage (Implicit)

- VUID-vkCreateFramebuffer-device-parameter
device **must** be a valid `VkDevice` handle
- VUID-vkCreateFramebuffer-pCreateInfo-parameter
pCreateInfo **must** be a valid pointer to a valid `VkFramebufferCreateInfo` structure
- VUID-vkCreateFramebuffer-pAllocator-null
pAllocator **must** be `NULL`
- VUID-vkCreateFramebuffer-pFramebuffer-parameter
pFramebuffer **must** be a valid pointer to a `VkFramebuffer` handle
- VUID-vkCreateFramebuffer-device-queuecount
The device **must** have been created with at least 1 queue

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkFramebufferCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkFramebufferCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkFramebufferCreateFlags flags;
    VkRenderPass          renderPass;
    uint32_t              attachmentCount;
    const VkImageView*    pAttachments;
    uint32_t              width;
    uint32_t              height;
    uint32_t              layers;
} VkFramebufferCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkFramebufferCreateFlagBits`
- `renderPass` is a render pass defining what render passes the framebuffer will be compatible with. See [Render Pass Compatibility](#) for details. The implementation **must** not access this object outside of the duration of the command this structure is passed to.
- `attachmentCount` is the number of attachments.
- `pAttachments` is a pointer to an array of `VkImageView` handles, each of which will be used as the corresponding attachment in a render pass instance. If `flags` includes `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, this parameter is ignored.
- `width`, `height` and `layers` define the dimensions of the framebuffer. If the render pass uses multiview, then `layers` **must** be one and each attachment requires a number of layers that is greater than the maximum bit index set in the view mask in the subpasses in which it is used.

For any depth/stencil attachments used by this framebuffer in `pAttachments`, the `aspectMask` is ignored.

It is legal for a subpass to use no color or depth/stencil attachments, either because it has no attachment references or because all of them are `VK_ATTACHMENT_UNUSED`. This kind of subpass **can** use shader side effects such as image stores and atomics to produce an output. In this case, the subpass continues to use the `width`, `height`, and `layers` of the framebuffer to define the dimensions of the rendering area, and the `rasterizationSamples` from each pipeline's `VkPipelineMultisampleStateCreateInfo` to define the number of samples used in rasterization; however, if `VkPhysicalDeviceFeatures::variableMultisampleRate` is `VK_FALSE`, then all pipelines to be bound with the subpass **must** have the same value for `VkPipelineMultisampleStateCreateInfo::rasterizationSamples`. In all such cases, `rasterizationSamples` **must** be a valid `VkSampleCountFlagBits` value that is set in `VkPhysicalDeviceLimits::framebufferNoAttachmentsSampleCounts`.

Valid Usage

- VUID-VkFramebufferCreateInfo-attachmentCount-00876

`attachmentCount` **must** be equal to the attachment count specified in `renderPass`

- VUID-VkFramebufferCreateInfo-flags-02778

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT` and `attachmentCount` is not 0, `pAttachments` **must** be a valid pointer to an array of `attachmentCount` valid `VkImageView` handles

- VUID-VkFramebufferCreateInfo-pAttachments-00877

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as a color attachment or resolve attachment by `renderPass` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`

- VUID-VkFramebufferCreateInfo-pAttachments-02633

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as a depth/stencil attachment by `renderPass` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-VkFramebufferCreateInfo-pAttachments-02634

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as a depth/stencil resolve attachment by `renderPass` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-VkFramebufferCreateInfo-pAttachments-00879

If `renderpass` is not `VK_NULL_HANDLE`, `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as an input attachment by `renderPass` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`

- VUID-VkFramebufferCreateInfo-pAttachments-00880

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` **must** have been created with a `VkFormat` value that matches the `VkFormat` specified by the corresponding `VkAttachmentDescription` in `renderPass`

- VUID-VkFramebufferCreateInfo-pAttachments-00881

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` **must** have been created with a `samples` value that matches the `samples` value specified by the corresponding `VkAttachmentDescription` in `renderPass`

- VUID-VkFramebufferCreateInfo-flags-04533

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as an input, color, resolve, or depth/stencil attachment by `renderPass` **must** have been created with a `VkImageCreateInfo::extent.width` greater than or equal to `width`

- VUID-VkFramebufferCreateInfo-flags-04534

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as an input, color, resolve, or depth/stencil attachment by `renderPass` **must** have been created with a `VkImageCreateInfo::extent.height` greater than or equal to `height`

- VUID-VkFramebufferCreateInfo-flags-04535

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as an input, color, resolve, or depth/stencil attachment by `renderPass` **must** have been created with a `VkImageViewCreateInfo`

`::subresourceRange.layerCount` greater than or equal to `layers`

- VUID-VkFramebufferCreateInfo-renderPass-04536

If `renderPass` was specified with non-zero view masks, each element of `pAttachments` that is used as an input, color, resolve, or depth/stencil attachment by `renderPass` **must** have a `layerCount` greater than the index of the most significant bit set in any of those view masks

- VUID-VkFramebufferCreateInfo-flags-04537

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, and `renderPass` was specified with non-zero view masks, each element of `pAttachments` that is used as a `fragment shading rate attachment` by `renderPass` **must** have a `layerCount` that is either 1, or greater than the index of the most significant bit set in any of those view masks

- VUID-VkFramebufferCreateInfo-flags-04538

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, and `renderPass` was not specified with non-zero view masks, each element of `pAttachments` that is used as a `fragment shading rate attachment` by `renderPass` **must** have a `layerCount` that is either 1, or greater than `layers`

- VUID-VkFramebufferCreateInfo-flags-04539

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, an element of `pAttachments` that is used as a `fragment shading rate attachment` **must** have a width at least as large as $\lceil \text{width} / \text{texelWidth} \rceil$, where `texelWidth` is the largest value of `shadingRateAttachmentTexelSize.width` in a `VkFragmentShadingRateAttachmentInfoKHR` which references that attachment

- VUID-VkFramebufferCreateInfo-flags-04540

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, an element of `pAttachments` that is used as a `fragment shading rate attachment` **must** have a height at least as large as $\lceil \text{height} / \text{texelHeight} \rceil$, where `texelHeight` is the largest value of `shadingRateAttachmentTexelSize.height` in a `VkFragmentShadingRateAttachmentInfoKHR` which references that attachment

- VUID-VkFramebufferCreateInfo-pAttachments-00883

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` **must** only specify a single mip level

- VUID-VkFramebufferCreateInfo-pAttachments-00884

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` **must** have been created with the identity swizzle

- VUID-VkFramebufferCreateInfo-width-00885

`width` **must** be greater than 0

- VUID-VkFramebufferCreateInfo-width-00886

`width` **must** be less than or equal to `maxFramebufferWidth`

- VUID-VkFramebufferCreateInfo-height-00887

`height` **must** be greater than 0

- VUID-VkFramebufferCreateInfo-height-00888

`height` **must** be less than or equal to `maxFramebufferHeight`

- VUID-VkFramebufferCreateInfo-layers-00889

`layers` **must** be greater than 0

- VUID-VkFramebufferCreateInfo-layers-00890
layers **must** be less than or equal to maxFramebufferLayers
- VUID-VkFramebufferCreateInfo-renderPass-02531
If renderPass was specified with non-zero view masks, layers **must** be 1
- VUID-VkFramebufferCreateInfo-pAttachments-00891
If flags does not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, each element of pAttachments that is a 2D or 2D array image view taken from a 3D image **must** not be a depth/stencil format
- VUID-VkFramebufferCreateInfo-flags-03189
If the imagelessFramebuffer feature is not enabled, flags **must** not include VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT
- VUID-VkFramebufferCreateInfo-flags-03190
If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the pNext chain **must** include a VkFramebufferAttachmentsCreateInfo structure
- VUID-VkFramebufferCreateInfo-flags-03191
If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the attachmentImageInfoCount member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain **must** be equal to either zero or attachmentCount
- VUID-VkFramebufferCreateInfo-flags-04541
If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the width member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain that is used as an input, color, resolve or depth/stencil attachment in renderPass **must** be greater than or equal to width
- VUID-VkFramebufferCreateInfo-flags-04542
If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the height member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain that is used as an input, color, resolve or depth/stencil attachment in renderPass **must** be greater than or equal to height
- VUID-VkFramebufferCreateInfo-flags-04543
If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the width member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain that is used as a fragment shading rate attachment **must** be greater than or equal to $\lceil \text{width} / \text{texelWidth} \rceil$, where texelWidth is the largest value of shadingRateAttachmentTexelSize.width in a VkFragmentShadingRateAttachmentInfoKHR which references that attachment
- VUID-VkFramebufferCreateInfo-flags-04544
If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the height member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure in the pNext chain that is used as a fragment shading rate attachment **must** be greater than or equal to $\lceil \text{height} / \text{texelHeight} \rceil$, where texelHeight is the largest value of shadingRateAttachmentTexelSize.height in a VkFragmentShadingRateAttachmentInfoKHR which references that attachment
- VUID-VkFramebufferCreateInfo-flags-04545
If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the layerCount member of any

element of the `pAttachmentImageInfos` member of a `VkFramebufferAttachmentsCreateInfo` structure in the `pNext` chain that is used as a `fragment shading rate attachment` **must** be either 1, or greater than or equal to `layers`

- VUID-VkFramebufferCreateInfo-flags-04587

If `flags` includes `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT` and `renderPass` was specified with non-zero view masks, the `layerCount` member of any element of the `pAttachmentImageInfos` member of a `VkFramebufferAttachmentsCreateInfo` structure in the `pNext` chain that is used as a `fragment shading rate attachment` **must** be either 1, or greater than the index of the most significant bit set in any of those view masks

- VUID-VkFramebufferCreateInfo-renderPass-03198

If `multiview` is enabled for `renderPass` and `flags` includes `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, the `layerCount` member of any element of the `pAttachmentImageInfos` member of a `VkFramebufferAttachmentsCreateInfo` structure included in the `pNext` chain used as an input, color, resolve, or depth/stencil attachment in `renderPass` **must** be greater than the maximum bit index set in the view mask in the subpasses in which it is used in `renderPass`

- VUID-VkFramebufferCreateInfo-renderPass-04546

If `multiview` is not enabled for `renderPass` and `flags` includes `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, the `layerCount` member of any element of the `pAttachmentImageInfos` member of a `VkFramebufferAttachmentsCreateInfo` structure included in the `pNext` chain used as an input, color, resolve, or depth/stencil attachment in `renderPass` **must** be greater than or equal to `layers`

- VUID-VkFramebufferCreateInfo-flags-03201

If `flags` includes `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, the `usage` member of any element of the `pAttachmentImageInfos` member of a `VkFramebufferAttachmentsCreateInfo` structure included in the `pNext` chain that refers to an attachment used as a color attachment or resolve attachment by `renderPass` **must** include `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`

- VUID-VkFramebufferCreateInfo-flags-03202

If `flags` includes `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, the `usage` member of any element of the `pAttachmentImageInfos` member of a `VkFramebufferAttachmentsCreateInfo` structure included in the `pNext` chain that refers to an attachment used as a depth/stencil attachment by `renderPass` **must** include `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-VkFramebufferCreateInfo-flags-03204

If `flags` includes `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, the `usage` member of any element of the `pAttachmentImageInfos` member of a `VkFramebufferAttachmentsCreateInfo` structure included in the `pNext` chain that refers to an attachment used as an input attachment by `renderPass` **must** include `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`

- VUID-VkFramebufferCreateInfo-flags-03205

If `flags` includes `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, at least one element of the `pViewFormats` member of any element of the `pAttachmentImageInfos` member of a `VkFramebufferAttachmentsCreateInfo` structure included in the `pNext` chain **must** be equal to the corresponding value of `VkAttachmentDescription::format` used to create `renderPass`

- VUID-VkFramebufferCreateInfo-flags-04113

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` **must** have been created with `VkImageViewCreateInfo::viewType` not equal to `VK_IMAGE_VIEW_TYPE_3D`

- VUID-VkFramebufferCreateInfo-flags-04548

If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as a fragment shading rate attachment by `renderPass` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`

- VUID-VkFramebufferCreateInfo-flags-04549

If `flags` includes `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, the `usage` member of any element of the `pAttachmentImageInfos` member of a `VkFramebufferAttachmentsCreateInfo` structure included in the `pNext` chain that refers to an attachment used as a fragment shading rate attachment by `renderPass` **must** include `VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`

- VUID-VkFramebufferCreateInfo-attachmentCount-05060

`attachmentCount` **must** be less than or equal to `maxFramebufferAttachments`

Valid Usage (Implicit)

- VUID-VkFramebufferCreateInfo-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO`

- VUID-VkFramebufferCreateInfo-pNext-pNext

`pNext` **must** be `NULL` or a pointer to a valid instance of `VkFramebufferAttachmentsCreateInfo`

- VUID-VkFramebufferCreateInfo-sType-unique

The `sType` value of each structure in the `pNext` chain **must** be unique

- VUID-VkFramebufferCreateInfo-flags-parameter

`flags` **must** be a valid combination of `VkFramebufferCreateFlagBits` values

- VUID-VkFramebufferCreateInfo-renderPass-parameter

`renderPass` **must** be a valid `VkRenderPass` handle

- VUID-VkFramebufferCreateInfo-commonparent

Both of `renderPass`, and the elements of `pAttachments` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same `VkDevice`

The `VkFramebufferAttachmentsCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkFramebufferAttachmentsCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    uint32_t                 attachmentImageInfoCount;
    const VkFramebufferAttachmentImageInfo* pAttachmentImageInfos;
} VkFramebufferAttachmentsCreateInfo;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **attachmentImageInfoCount** is the number of attachments being described.
- **pAttachmentImageInfos** is a pointer to an array of [VkFramebufferAttachmentImageInfo](#) structures, each structure describing a number of parameters of the corresponding attachment in a render pass instance.

Valid Usage (Implicit)

- VUID-VkFramebufferAttachmentsCreateInfo-sType-sType
sType must be **VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENTS_CREATE_INFO**
- VUID-VkFramebufferAttachmentsCreateInfo-pAttachmentImageInfos-parameter
If **attachmentImageInfoCount** is not 0, **pAttachmentImageInfos must** be a valid pointer to an array of **attachmentImageInfoCount** valid [VkFramebufferAttachmentImageInfo](#) structures

The [VkFramebufferAttachmentImageInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkFramebufferAttachmentImageInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkImageCreateFlags        flags;
    VkImageUsageFlags         usage;
    uint32_t                 width;
    uint32_t                 height;
    uint32_t                 layerCount;
    uint32_t                 viewFormatCount;
    const VkFormat*          pViewFormats;
} VkFramebufferAttachmentImageInfo;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **flags** is a bitmask of [VkImageCreateFlagBits](#), matching the value of [VkImageCreateInfo::flags](#) used to create an image that will be used with this framebuffer.
- **usage** is a bitmask of [VkImageUsageFlagBits](#), matching the value of [VkImageCreateInfo::usage](#)

used to create an image used with this framebuffer.

- **width** is the width of the image view used for rendering.
- **height** is the height of the image view used for rendering.
- **layerCount** is the number of array layers of the image view used for rendering.
- **viewFormatCount** is the number of entries in the **pViewFormats** array, matching the value of **VkImageFormatListCreateInfo::viewFormatCount** used to create an image used with this framebuffer.
- **pViewFormats** is a pointer to an array of **VkFormat** values specifying all of the formats which **can** be used when creating views of the image, matching the value of **VkImageFormatListCreateInfo::pViewFormats** used to create an image used with this framebuffer.

Images that **can** be used with the framebuffer when beginning a render pass, as specified by **VkRenderPassAttachmentBeginInfo**, **must** be created with parameters that are identical to those specified here.

Valid Usage

- VUID-VkFramebufferAttachmentImageInfo-viewFormatCount-09536
If **viewFormatCount** is not 0, each element of **pViewFormats** **must** not be **VK_FORMAT_UNDEFINED**

Valid Usage (Implicit)

- VUID-VkFramebufferAttachmentImageInfo-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENT_IMAGE_INFO**
- VUID-VkFramebufferAttachmentImageInfo-pNext-pNext
pNext **must** be **NULL**
- VUID-VkFramebufferAttachmentImageInfo-flags-parameter
flags **must** be a valid combination of **VkImageCreateFlagBits** values
- VUID-VkFramebufferAttachmentImageInfo-usage-parameter
usage **must** be a valid combination of **VkImageUsageFlagBits** values
- VUID-VkFramebufferAttachmentImageInfo-usage-requiredbitmask
usage **must** not be 0
- VUID-VkFramebufferAttachmentImageInfo-pViewFormats-parameter
If **viewFormatCount** is not 0, **pViewFormats** **must** be a valid pointer to an array of **viewFormatCount** valid **VkFormat** values

Bits which **can** be set in **VkFramebufferCreateInfo::flags**, specifying options for framebuffers, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkFramebufferCreateFlagBits {
    // Provided by VK_VERSION_1_2
    VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT = 0x00000001,
} VkFramebufferCreateFlagBits;
```

- `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT` specifies that image views are not specified, and only attachment compatibility information will be provided via a `VkFramebufferAttachmentImageInfo` structure.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkFramebufferCreateFlags;
```

`VkFramebufferCreateFlags` is a bitmask type for setting a mask of zero or more `VkFramebufferCreateFlagBits`.

To destroy a framebuffer, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyFramebuffer(
    VkDevice          device,
    VkFramebuffer     framebuffer,
    const VkAllocationCallbacks* pAllocator);
```

- `device` is the logical device that destroys the framebuffer.
- `framebuffer` is the handle of the framebuffer to destroy.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage

- VUID-vkDestroyFramebuffer-framebuffer-00892
All submitted commands that refer to `framebuffer` **must** have completed execution

Valid Usage (Implicit)

- VUID-vkDestroyFramebuffer-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkDestroyFramebuffer-framebuffer-parameter
If `framebuffer` is not `VK_NULL_HANDLE`, `framebuffer` **must** be a valid `VkFramebuffer` handle
- VUID-vkDestroyFramebuffer-pAllocator-null
`pAllocator` **must** be `NULL`

- VUID-vkDestroyFramebuffer-framebuffer-parent

If `framebuffer` is a valid handle, it **must** have been created, allocated, or retrieved from device

Host Synchronization

- Host access to `framebuffer` **must** be externally synchronized

8.4. Render Pass Load Operations

Render pass load operations define the initial values of an attachment during a render pass instance.

Load operations for attachments with a depth/stencil format execute in the `VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT` pipeline stage. Load operations for attachments with a color format execute in the `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT` pipeline stage. The load operation for each sample in an attachment happens-before any recorded command which accesses the sample in that render pass instance via that attachment or an alias. In a render pass object with multiple `subpasses`, load operations are performed in the first subpass which uses an attachment.

NOTE

Because load operations always happen first, external synchronization with attachment access only needs to synchronize the load operations with previous commands; not the operations within the render pass instance.

Load operations only update values within the defined render area for the render pass instance. However, any writes performed by a load operation (as defined by its access masks) to a given attachment **may** read and write back any memory locations within the image subresource bound for that attachment. For depth/stencil images, writes to one aspect **may** also result in read-modify-write operations for the other aspect.

NOTE

As entire subresources could be accessed by load operations, applications cannot safely access values outside of the render area during a render pass instance when a load operation that modifies values is used.

Load operations that **can** be used for a render pass are:

```
// Provided by VK_VERSION_1_0
typedef enum VkAttachmentLoadOp {
    VK_ATTACHMENT_LOAD_OP_LOAD = 0,
    VK_ATTACHMENT_LOAD_OP_CLEAR = 1,
    VK_ATTACHMENT_LOAD_OP_DONT_CARE = 2,
} VkAttachmentLoadOp;
```

- `VK_ATTACHMENT_LOAD_OP_LOAD` specifies that the previous contents of the image within the render

area will be preserved as the initial values. For attachments with a depth/stencil format, this uses the access type `VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT`. For attachments with a color format, this uses the access type `VK_ACCESS_COLOR_ATTACHMENT_READ_BIT`.

- `VK_ATTACHMENT_LOAD_OP_CLEAR` specifies that the contents within the render area will be cleared to a uniform value, which is specified when a render pass instance is begun. For attachments with a depth/stencil format, this uses the access type `VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`. For attachments with a color format, this uses the access type `VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT`.
- `VK_ATTACHMENT_LOAD_OP_DONT_CARE` specifies that the previous contents within the area need not be preserved; the contents of the attachment will be undefined inside the render area. For attachments with a depth/stencil format, this uses the access type `VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`. For attachments with a color format, this uses the access type `VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT`.

During a render pass instance, input and color attachments with color formats that have a component size of 8, 16, or 32 bits **must** be represented in the attachment's format throughout the instance. Attachments with other floating- or fixed-point color formats, or with depth components **may** be represented in a format with a precision higher than the attachment format, but **must** be represented with the same range. When such a component is loaded via the `loadOp`, it will be converted into an implementation-dependent format used by the render pass. Such components **must** be converted from the render pass format, to the format of the attachment, before they are resolved or stored at the end of a render pass instance via `storeOp`. Conversions occur as described in [Numeric Representation and Computation](#) and [Fixed-Point Data Conversions](#).

8.5. Render Pass Store Operations

Render pass store operations define how values written to an attachment during a render pass instance are stored to memory.

Store operations for attachments with a depth/stencil format execute in the `VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT` pipeline stage. Store operations for attachments with a color format execute in the `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT` pipeline stage. The store operation for each sample in an attachment happens-after any recorded command which accesses the sample via that attachment or an alias. In a render pass object with multiple [subpasses](#), store operations are performed in the last subpass which uses an attachment.

NOTE

Because store operations always happen after other accesses in a render pass instance, external synchronization with attachment access in an earlier render pass only needs to synchronize with the store operations; not the operations within the render pass instance.

Store operations only update values within the defined render area for the render pass instance. However, any writes performed by a store operation (as defined by its access masks) to a given attachment **may** read and write back any memory locations within the image subresource bound for that attachment. For depth/stencil images, writes to one aspect **may** also result in read-modify-write operations for the other aspect.

NOTE

As entire subresources could be accessed by store operations, applications cannot safely access values outside of the render area via aliased resources during a render pass instance when a store operation that modifies values is used.

Possible values of `VkAttachmentDescription::storeOp` and `stencilStoreOp`, specifying how the contents of the attachment are treated, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkAttachmentStoreOp {
    VK_ATTACHMENT_STORE_OP_STORE = 0,
    VK_ATTACHMENT_STORE_OP_DONT_CARE = 1,
} VkAttachmentStoreOp;
```

- `VK_ATTACHMENT_STORE_OP_STORE` specifies the contents generated during the render pass and within the render area are written to memory. For attachments with a depth/stencil format, this uses the access type `VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`. For attachments with a color format, this uses the access type `VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT`.
- `VK_ATTACHMENT_STORE_OP_DONT_CARE` specifies the contents within the render area are not needed after rendering, and **may** be discarded; the contents of the attachment will be undefined inside the render area. For attachments with a depth/stencil format, this uses the access type `VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`. For attachments with a color format, this uses the access type `VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT`.

NOTE

`VK_ATTACHMENT_STORE_OP_DONT_CARE` **can** cause contents generated during previous render passes to be discarded before reaching memory, even if no write to the attachment occurs during the current render pass.

8.6. Render Pass Multisample Resolve Operations

Render pass multisample resolve operations combine sample values from a single pixel in a multisample attachment and store the result to the corresponding pixel in a single sample attachment.

Multisample resolve operations for attachments execute in the `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT` pipeline stage. A final resolve operation for all pixels in the render area happens-after any recorded command which writes a pixel via the multisample attachment to be resolved or an explicit alias of it in the subpass that it is specified. Any single sample attachment specified for use in a multisample resolve operation **may** have its contents modified at any point once rendering begins for the render pass instance. Reads from the multisample attachment can be synchronized with `VK_ACCESS_COLOR_ATTACHMENT_READ_BIT`. Access to the single sample attachment can be synchronized with `VK_ACCESS_COLOR_ATTACHMENT_READ_BIT` and `VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT`. These pipeline stage and access types are used whether the attachments are color or depth/stencil attachments.

When using render pass objects, a subpass dependency specified with the above pipeline stages and access flags will ensure synchronization with multisample resolve operations for any

attachments that were last accessed by that subpass. This allows later subpasses to read resolved values as input attachments.

Resolve operations only update values within the defined render area for the render pass instance. However, any writes performed by a resolve operation (as defined by its access masks) to a given attachment **may** read and write back any memory locations within the image subresource bound for that attachment. For depth/stencil images, writes to one aspect **may** also result in read-modify-write operations for the other aspect.

NOTE

As entire subresources could be accessed by multisample resolve operations, applications cannot safely access values outside of the render area via aliased resources during a render pass instance when a multisample resolve operation is performed.

Multisample values in a multisample attachment are combined according to the resolve mode used:

```
// Provided by VK_VERSION_1_2
typedef enum VkResolveModeFlagBits {
    VK_RESOLVE_MODE_NONE = 0,
    VK_RESOLVE_MODE_SAMPLE_ZERO_BIT = 0x00000001,
    VK_RESOLVE_MODE_AVERAGE_BIT = 0x00000002,
    VK_RESOLVE_MODE_MIN_BIT = 0x00000004,
    VK_RESOLVE_MODE_MAX_BIT = 0x00000008,
} VkResolveModeFlagBits;
```

- `VK_RESOLVE_MODE_NONE` specifies that no resolve operation is done.
- `VK_RESOLVE_MODE_SAMPLE_ZERO_BIT` specifies that result of the resolve operation is equal to the value of sample 0.
- `VK_RESOLVE_MODE_AVERAGE_BIT` specifies that result of the resolve operation is the average of the sample values.
- `VK_RESOLVE_MODE_MIN_BIT` specifies that result of the resolve operation is the minimum of the sample values.
- `VK_RESOLVE_MODE_MAX_BIT` specifies that result of the resolve operation is the maximum of the sample values.

If no resolve mode is otherwise specified, `VK_RESOLVE_MODE_AVERAGE_BIT` is used.

If `VK_RESOLVE_MODE_AVERAGE_BIT` is used, and the source format is a floating-point or normalized type, the sample values for each pixel are resolved with implementation-defined numerical precision.

If the [numeric format](#) of the resolve attachment uses sRGB encoding, the implementation **should** convert samples from nonlinear to linear before averaging samples as described in the “sRGB EOTF” section of the [Khronos Data Format Specification](#). In this case, the implementation **must** convert the linear averaged value to nonlinear before writing the resolved result to resolve attachment.

```
// Provided by VK_VERSION_1_2
typedef VkFlags VkResolveModeFlags;
```

`VkResolveModeFlags` is a bitmask type for setting a mask of zero or more `VkResolveModeFlagBits`.

8.7. Render Pass Commands

An application records the commands for a render pass instance one subpass at a time, by beginning a render pass instance, iterating over the subpasses to record commands for that subpass, and then ending the render pass instance.

To begin a render pass instance, call:

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
void vkCmdBeginRenderPass(
    VkCommandBuffer                commandBuffer,
    const VkRenderPassBeginInfo*   pRenderPassBegin,
    VkSubpassContents              contents);
```

- `commandBuffer` is the command buffer in which to record the command.
- `pRenderPassBegin` is a pointer to a `VkRenderPassBeginInfo` structure specifying the render pass to begin an instance of, and the framebuffer the instance uses.
- `contents` is a `VkSubpassContents` value specifying how the commands in the first subpass will be provided.

After beginning a render pass instance, the command buffer is ready to record the commands for the first subpass of that render pass.

Valid Usage

- VUID-vkCmdBeginRenderPass-initialLayout-00895
If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
- VUID-vkCmdBeginRenderPass-initialLayout-01758
If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is

VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL,
VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL,
VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL, or
VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-vkCmdBeginRenderPass-initialLayout-02842

If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-vkCmdBeginRenderPass-stencilInitialLayout-02843

If any of the `stencilInitialLayout` or `stencilFinalLayout` member of the `VkAttachmentDescriptionStencilLayout` structures or the `stencilLayout` member of the `VkAttachmentReferenceStencilLayout` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-vkCmdBeginRenderPass-initialLayout-00897

If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_SAMPLED_BIT` or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`

- VUID-vkCmdBeginRenderPass-initialLayout-00898

If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_TRANSFER_SRC_BIT`

- VUID-vkCmdBeginRenderPass-initialLayout-00899

If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is

`VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_TRANSFER_DST_BIT`

- VUID-vkCmdBeginRenderPass-initialLayout-00900

If the `initialLayout` member of any of the `VkAttachmentDescription` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is not `VK_IMAGE_LAYOUT_UNDEFINED`, then each such `initialLayout` **must** be equal to the current layout of the corresponding attachment image subresource of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin`

- VUID-vkCmdBeginRenderPass-srcStageMask-06451

The `srcStageMask` members of any element of the `pDependencies` member of `VkRenderPassCreateInfo` used to create `renderPass` **must** be supported by the capabilities of the queue family identified by the `queueFamilyIndex` member of the `VkCommandPoolCreateInfo` used to create the command pool which `commandBuffer` was allocated from

- VUID-vkCmdBeginRenderPass-dstStageMask-06452

The `dstStageMask` members of any element of the `pDependencies` member of `VkRenderPassCreateInfo` used to create `renderPass` **must** be supported by the capabilities of the queue family identified by the `queueFamilyIndex` member of the `VkCommandPoolCreateInfo` used to create the command pool which `commandBuffer` was allocated from

- VUID-vkCmdBeginRenderPass-framebuffer-02532

For any attachment in `framebuffer` that is used by `renderPass` and is bound to memory locations that are also bound to another attachment used by `renderPass`, and if at least one of those uses causes either attachment to be written to, both attachments **must** have had the `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT` set

- VUID-vkCmdBeginRenderPass-framebuffer-09045

If any attachments specified in `framebuffer` are used by `renderPass` and are bound to overlapping memory locations, there **must** be only one that is used as a color attachment, depth/stencil, or resolve attachment in any subpass

Valid Usage (Implicit)

- VUID-vkCmdBeginRenderPass-commandBuffer-parameter

`commandBuffer` **must** be a valid `VkCommandBuffer` handle

- VUID-vkCmdBeginRenderPass-pRenderPassBegin-parameter

`pRenderPassBegin` **must** be a valid pointer to a valid `VkRenderPassBeginInfo` structure

- VUID-vkCmdBeginRenderPass-contents-parameter

`contents` **must** be a valid `VkSubpassContents` value

- VUID-vkCmdBeginRenderPass-commandBuffer-recording

`commandBuffer` **must** be in the `recording` state

- VUID-vkCmdBeginRenderPass-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support

VK_QUEUE_GRAPHICS_BIT operations

- VUID-vkCmdBeginRenderPass-renderpass

This command **must** only be called outside of a render pass instance

- VUID-vkCmdBeginRenderPass-bufferlevel
commandBuffer **must** be a primary VkCommandBuffer

Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary	Outside	VK_QUEUE_GRAPHICS_BIT	Action State Synchronization

Alternatively to begin a render pass, call:

```
// Provided by VK_VERSION_1_2
void vkCmdBeginRenderPass2(
    VkCommandBuffer                commandBuffer,
    const VkRenderPassBeginInfo*   pRenderPassBegin,
    const VkSubpassBeginInfo*      pSubpassBeginInfo);
```

- commandBuffer is the command buffer in which to record the command.
- pRenderPassBegin is a pointer to a VkRenderPassBeginInfo structure specifying the render pass to begin an instance of, and the framebuffer the instance uses. After recording this command, the render pass and framebuffer **may** be accessed at any point that commandBuffer is in the recording or pending state until it is reset.
- pSubpassBeginInfo is a pointer to a VkSubpassBeginInfo structure containing information about the subpass which is about to begin rendering.

After beginning a render pass instance, the command buffer is ready to record the commands for the first subpass of that render pass.

Valid Usage

- VUID-vkCmdBeginRenderPass2-framebuffer-02779

Both the `framebuffer` and `renderPass` members of `pRenderPassBegin` **must** have been created on the same `VkDevice` that `commandBuffer` was allocated on

- VUID-vkCmdBeginRenderPass2-initialLayout-03094

If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`

- VUID-vkCmdBeginRenderPass2-initialLayout-03096

If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-vkCmdBeginRenderPass2-initialLayout-02844

If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-vkCmdBeginRenderPass2-stencilInitialLayout-02845

If any of the `stencilInitialLayout` or `stencilFinalLayout` member of the `VkAttachmentDescriptionStencilLayout` structures or the `stencilLayout` member of the `VkAttachmentReferenceStencilLayout` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-vkCmdBeginRenderPass2-initialLayout-03097

If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have

been created with a `usage` value including `VK_IMAGE_USAGE_SAMPLED_BIT` or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`

- VUID-vkCmdBeginRenderPass2-initialLayout-03098

If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_TRANSFER_SRC_BIT`

- VUID-vkCmdBeginRenderPass2-initialLayout-03099

If any of the `initialLayout` or `finalLayout` member of the `VkAttachmentDescription` structures or the `layout` member of the `VkAttachmentReference` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` then the corresponding attachment image view of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin` **must** have been created with a `usage` value including `VK_IMAGE_USAGE_TRANSFER_DST_BIT`

- VUID-vkCmdBeginRenderPass2-initialLayout-03100

If the `initialLayout` member of any of the `VkAttachmentDescription` structures specified when creating the render pass specified in the `renderPass` member of `pRenderPassBegin` is not `VK_IMAGE_LAYOUT_UNDEFINED`, then each such `initialLayout` **must** be equal to the current layout of the corresponding attachment image subresource of the framebuffer specified in the `framebuffer` member of `pRenderPassBegin`

- VUID-vkCmdBeginRenderPass2-srcStageMask-06453

The `srcStageMask` members of any element of the `pDependencies` member of `VkRenderPassCreateInfo` used to create `renderPass` **must** be supported by the capabilities of the queue family identified by the `queueFamilyIndex` member of the `VkCommandPoolCreateInfo` used to create the command pool which `commandBuffer` was allocated from

- VUID-vkCmdBeginRenderPass2-dstStageMask-06454

The `dstStageMask` members of any element of the `pDependencies` member of `VkRenderPassCreateInfo` used to create `renderPass` **must** be supported by the capabilities of the queue family identified by the `queueFamilyIndex` member of the `VkCommandPoolCreateInfo` used to create the command pool which `commandBuffer` was allocated from

- VUID-vkCmdBeginRenderPass2-framebuffer-02533

For any attachment in `framebuffer` that is used by `renderPass` and is bound to memory locations that are also bound to another attachment used by `renderPass`, and if at least one of those uses causes either attachment to be written to, both attachments **must** have had the `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT` set

- VUID-vkCmdBeginRenderPass2-framebuffer-09046

If any attachments specified in `framebuffer` are used by `renderPass` and are bound to overlapping memory locations, there **must** be only one that is used as a color attachment, depth/stencil, or resolve attachment in any subpass

Valid Usage (Implicit)

- VUID-vkCmdBeginRenderPass2-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdBeginRenderPass2-pRenderPassBegin-parameter
`pRenderPassBegin` **must** be a valid pointer to a valid `VkRenderPassBeginInfo` structure
- VUID-vkCmdBeginRenderPass2-pSubpassBeginInfo-parameter
`pSubpassBeginInfo` **must** be a valid pointer to a valid `VkSubpassBeginInfo` structure
- VUID-vkCmdBeginRenderPass2-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdBeginRenderPass2-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdBeginRenderPass2-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdBeginRenderPass2-bufferlevel
`commandBuffer` **must** be a primary `VkCommandBuffer`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary	Outside	VK_QUEUE_GRAPHICS_BIT	Action State Synchronization

The `VkRenderPassBeginInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkRenderPassBeginInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkRenderPass        renderPass;
    VkFramebuffer       framebuffer;
    VkRect2D            renderArea;
    uint32_t            clearValueCount;
    const VkClearColor* pClearValues;
} VkRenderPassBeginInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `renderPass` is the render pass to begin an instance of.
- `framebuffer` is the framebuffer containing the attachments that are used with the render pass.
- `renderArea` is the render area that is affected by the render pass instance, and is described in more detail below.
- `clearValueCount` is the number of elements in `pClearValues`.
- `pClearValues` is a pointer to an array of `clearValueCount` `VkClearColor` structures containing clear values for each attachment, if the attachment uses a `loadOp` value of `VK_ATTACHMENT_LOAD_OP_CLEAR` or if the attachment has a depth/stencil format and uses a `stencilLoadOp` value of `VK_ATTACHMENT_LOAD_OP_CLEAR`. The array is indexed by attachment number. Only elements corresponding to cleared attachments are used. Other elements of `pClearValues` are ignored.

`renderArea` is the render area that is affected by the render pass instance. The effects of attachment load, store and multisample resolve operations are restricted to the pixels whose x and y coordinates fall within the render area on all attachments. The render area extends to all layers of `framebuffer`. The application **must** ensure (using scissor if necessary) that all rendering is contained within the render area. The render area **must** be contained within the framebuffer dimensions.

NOTE There **may** be a performance cost for using a render area smaller than the framebuffer, unless it matches the render area granularity for the render pass.

Valid Usage

- VUID-VkRenderPassBeginInfo-clearValueCount-00902
`clearValueCount` **must** be greater than the largest attachment index in `renderPass` specifying a `loadOp` (or `stencilLoadOp`, if the attachment has a depth/stencil format) of `VK_ATTACHMENT_LOAD_OP_CLEAR`
- VUID-VkRenderPassBeginInfo-clearValueCount-04962
If `clearValueCount` is not 0, `pClearValues` **must** be a valid pointer to an array of `clearValueCount` `VkClearColor` unions
- VUID-VkRenderPassBeginInfo-renderPass-00904

`renderPass` **must** be `compatible` with the `renderPass` member of the `VkFramebufferCreateInfo` structure specified when creating `framebuffer`

- VUID-VkRenderPassBeginInfo-None-08996
If the `pNext` chain does not contain `VkDeviceGroupRenderPassBeginInfo` or its `deviceRenderAreaCount` member is equal to 0, `renderArea.extent.width` **must** be greater than 0
- VUID-VkRenderPassBeginInfo-None-08997
If the `pNext` chain does not contain `VkDeviceGroupRenderPassBeginInfo` or its `deviceRenderAreaCount` member is equal to 0, `renderArea.extent.height` **must** be greater than 0
- VUID-VkRenderPassBeginInfo-pNext-02850
If the `pNext` chain does not contain `VkDeviceGroupRenderPassBeginInfo` or its `deviceRenderAreaCount` member is equal to 0, `renderArea.offset.x` **must** be greater than or equal to 0
- VUID-VkRenderPassBeginInfo-pNext-02851
If the `pNext` chain does not contain `VkDeviceGroupRenderPassBeginInfo` or its `deviceRenderAreaCount` member is equal to 0, `renderArea.offset.y` **must** be greater than or equal to 0
- VUID-VkRenderPassBeginInfo-pNext-02852
If the `pNext` chain does not contain `VkDeviceGroupRenderPassBeginInfo` or its `deviceRenderAreaCount` member is equal to 0, `renderArea.offset.x` + `renderArea.extent.width` **must** be less than or equal to `VkFramebufferCreateInfo::width` the `framebuffer` was created with
- VUID-VkRenderPassBeginInfo-pNext-02853
If the `pNext` chain does not contain `VkDeviceGroupRenderPassBeginInfo` or its `deviceRenderAreaCount` member is equal to 0, `renderArea.offset.y` + `renderArea.extent.height` **must** be less than or equal to `VkFramebufferCreateInfo::height` the `framebuffer` was created with
- VUID-VkRenderPassBeginInfo-pNext-02856
If the `pNext` chain contains `VkDeviceGroupRenderPassBeginInfo`, `offset.x` + `extent.width` of each element of `pDeviceRenderAreas` **must** be less than or equal to `VkFramebufferCreateInfo::width` the `framebuffer` was created with
- VUID-VkRenderPassBeginInfo-pNext-02857
If the `pNext` chain contains `VkDeviceGroupRenderPassBeginInfo`, `offset.y` + `extent.height` of each element of `pDeviceRenderAreas` **must** be less than or equal to `VkFramebufferCreateInfo::height` the `framebuffer` was created with
- VUID-VkRenderPassBeginInfo-framebuffer-03207
If `framebuffer` was created with a `VkFramebufferCreateInfo::flags` value that did not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, and the `pNext` chain includes a `VkRenderPassAttachmentBeginInfo` structure, its `attachmentCount` **must** be zero
- VUID-VkRenderPassBeginInfo-framebuffer-03208
If `framebuffer` was created with a `VkFramebufferCreateInfo::flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, the `attachmentCount` of a `VkRenderPassAttachmentBeginInfo` structure included in the `pNext` chain **must** be equal

to the value of `VkFramebufferAttachmentsCreateInfo::attachmentImageInfoCount` used to create `framebuffer`

- VUID-VkRenderPassBeginInfo-framebuffer-02780

If `framebuffer` was created with a `VkFramebufferCreateInfo::flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of the `pAttachments` member of a `VkRenderPassAttachmentBeginInfo` structure included in the `pNext` chain **must** have been created on the same `VkDevice` as `framebuffer` and `renderPass`

- VUID-VkRenderPassBeginInfo-framebuffer-03209

If `framebuffer` was created with a `VkFramebufferCreateInfo::flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of the `pAttachments` member of a `VkRenderPassAttachmentBeginInfo` structure included in the `pNext` chain **must** be a `VkImageView` of an image created with a value of `VkImageCreateInfo::flags` equal to the `flags` member of the corresponding element of `VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos` used to create `framebuffer`

- VUID-VkRenderPassBeginInfo-framebuffer-04627

If `framebuffer` was created with a `VkFramebufferCreateInfo::flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of the `pAttachments` member of a `VkRenderPassAttachmentBeginInfo` structure included in the `pNext` chain **must** be a `VkImageView` with an `inherited usage` equal to the `usage` member of the corresponding element of `VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos` used to create `framebuffer`

- VUID-VkRenderPassBeginInfo-framebuffer-03211

If `framebuffer` was created with a `VkFramebufferCreateInfo::flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of the `pAttachments` member of a `VkRenderPassAttachmentBeginInfo` structure included in the `pNext` chain **must** be a `VkImageView` with a `width` equal to the `width` member of the corresponding element of `VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos` used to create `framebuffer`

- VUID-VkRenderPassBeginInfo-framebuffer-03212

If `framebuffer` was created with a `VkFramebufferCreateInfo::flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of the `pAttachments` member of a `VkRenderPassAttachmentBeginInfo` structure included in the `pNext` chain **must** be a `VkImageView` with a `height` equal to the `height` member of the corresponding element of `VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos` used to create `framebuffer`

- VUID-VkRenderPassBeginInfo-framebuffer-03213

If `framebuffer` was created with a `VkFramebufferCreateInfo::flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of the `pAttachments` member of a `VkRenderPassAttachmentBeginInfo` structure included in the `pNext` chain **must** be a `VkImageView` of an image created with a value of `VkImageViewCreateInfo::subresourceRange.layerCount` equal to the `layerCount` member of the corresponding element of `VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos` used to create `framebuffer`

- VUID-VkRenderPassBeginInfo-framebuffer-03214

If `framebuffer` was created with a `VkFramebufferCreateInfo::flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of the `pAttachments` member of a `VkRenderPassAttachmentBeginInfo` structure included in the `pNext` chain **must** be a

`VkImageView` of an image created with a value of `VkImageFormatListCreateInfo::viewFormatCount` equal to the `viewFormatCount` member of the corresponding element of `VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos` used to create `framebuffer`

- VUID-VkRenderPassBeginInfo-framebuffer-03215

If `framebuffer` was created with a `VkFramebufferCreateInfo::flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of the `pAttachments` member of a `VkRenderPassAttachmentBeginInfo` structure included in the `pNext` chain **must** be a `VkImageView` of an image created with a set of elements in `VkImageFormatListCreateInfo::pViewFormats` equal to the set of elements in the `pViewFormats` member of the corresponding element of `VkFramebufferAttachmentsCreateInfo::pAttachmentImageInfos` used to create `framebuffer`

- VUID-VkRenderPassBeginInfo-framebuffer-03216

If `framebuffer` was created with a `VkFramebufferCreateInfo::flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of the `pAttachments` member of a `VkRenderPassAttachmentBeginInfo` structure included in the `pNext` chain **must** be a `VkImageView` of an image created with a value of `VkImageViewCreateInfo::format` equal to the corresponding value of `VkAttachmentDescription::format` in `renderPass`

- VUID-VkRenderPassBeginInfo-framebuffer-09047

If `framebuffer` was created with a `VkFramebufferCreateInfo::flags` value that included `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of the `pAttachments` member of a `VkRenderPassAttachmentBeginInfo` structure included in the `pNext` chain **must** be a `VkImageView` of an image created with a value of `VkImageCreateInfo::samples` equal to the corresponding value of `VkAttachmentDescription::samples` in `renderPass`

Valid Usage (Implicit)

- VUID-VkRenderPassBeginInfo-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO`

- VUID-VkRenderPassBeginInfo-pNext-pNext

Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkDeviceGroupRenderPassBeginInfo`, `VkRenderPassAttachmentBeginInfo`, or `VkRenderPassSampleLocationsBeginInfoEXT`

- VUID-VkRenderPassBeginInfo-sType-unique

The `sType` value of each structure in the `pNext` chain **must** be unique

- VUID-VkRenderPassBeginInfo-renderPass-parameter

`renderPass` **must** be a valid `VkRenderPass` handle

- VUID-VkRenderPassBeginInfo-framebuffer-parameter

`framebuffer` **must** be a valid `VkFramebuffer` handle

- VUID-VkRenderPassBeginInfo-commonparent

Both of `framebuffer`, and `renderPass` **must** have been created, allocated, or retrieved from the same `VkDevice`

The image layout of the depth aspect of a depth/stencil attachment referring to an image created

with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` is dependent on the last sample locations used to render to the image subresource, thus preserving the contents of such depth/stencil attachments across subpass boundaries requires the application to specify these sample locations whenever a layout transition of the attachment **may** occur. This information **can** be provided by adding a `VkRenderPassSampleLocationsBeginInfoEXT` structure to the `pNext` chain of `VkRenderPassBeginInfo`.

The `VkRenderPassSampleLocationsBeginInfoEXT` structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkRenderPassSampleLocationsBeginInfoEXT {
    VkStructureType          sType;
    const void*              pNext;
    uint32_t                 attachmentInitialSampleLocationsCount;
    const VkAttachmentSampleLocationsEXT* pAttachmentInitialSampleLocations;
    uint32_t                 postSubpassSampleLocationsCount;
    const VkSubpassSampleLocationsEXT* pPostSubpassSampleLocations;
} VkRenderPassSampleLocationsBeginInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `attachmentInitialSampleLocationsCount` is the number of elements in the `pAttachmentInitialSampleLocations` array.
- `pAttachmentInitialSampleLocations` is a pointer to an array of `attachmentInitialSampleLocationsCount` `VkAttachmentSampleLocationsEXT` structures specifying the attachment indices and their corresponding sample location state. Each element of `pAttachmentInitialSampleLocations` **can** specify the sample location state to use in the automatic layout transition performed to transition a depth/stencil attachment from the initial layout of the attachment to the image layout specified for the attachment in the first subpass using it.
- `postSubpassSampleLocationsCount` is the number of elements in the `pPostSubpassSampleLocations` array.
- `pPostSubpassSampleLocations` is a pointer to an array of `postSubpassSampleLocationsCount` `VkSubpassSampleLocationsEXT` structures specifying the subpass indices and their corresponding sample location state. Each element of `pPostSubpassSampleLocations` **can** specify the sample location state to use in the automatic layout transition performed to transition the depth/stencil attachment used by the specified subpass to the image layout specified in a dependent subpass or to the final layout of the attachment in case the specified subpass is the last subpass using that attachment. In addition, if `VkPhysicalDeviceSampleLocationsPropertiesEXT::variableSampleLocations` is `VK_FALSE`, each element of `pPostSubpassSampleLocations` **must** specify the sample location state that matches the sample locations used by all pipelines that will be bound to a command buffer during the specified subpass. If `variableSampleLocations` is `VK_TRUE`, the sample locations used for rasterization do not depend on `pPostSubpassSampleLocations`.

Valid Usage (Implicit)

- VUID-VkRenderPassSampleLocationsBeginInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_SAMPLE_LOCATIONS_BEGIN_INFO_EXT`
- VUID-VkRenderPassSampleLocationsBeginInfoEXT-pAttachmentInitialSampleLocations-parameter
If `attachmentInitialSampleLocationsCount` is not 0, `pAttachmentInitialSampleLocations` **must** be a valid pointer to an array of `attachmentInitialSampleLocationsCount` valid `VkAttachmentSampleLocationsEXT` structures
- VUID-VkRenderPassSampleLocationsBeginInfoEXT-pPostSubpassSampleLocations-parameter
If `postSubpassSampleLocationsCount` is not 0, `pPostSubpassSampleLocations` **must** be a valid pointer to an array of `postSubpassSampleLocationsCount` valid `VkSubpassSampleLocationsEXT` structures

The `VkAttachmentSampleLocationsEXT` structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkAttachmentSampleLocationsEXT {
    uint32_t          attachmentIndex;
    VkSampleLocationsInfoEXT sampleLocationsInfo;
} VkAttachmentSampleLocationsEXT;
```

- `attachmentIndex` is the index of the attachment for which the sample locations state is provided.
- `sampleLocationsInfo` is the sample locations state to use for the layout transition of the given attachment from the initial layout of the attachment to the image layout specified for the attachment in the first subpass using it.

If the image referenced by the framebuffer attachment at index `attachmentIndex` was not created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` then the values specified in `sampleLocationsInfo` are ignored.

Valid Usage

- VUID-VkAttachmentSampleLocationsEXT-attachmentIndex-01531
`attachmentIndex` **must** be less than the `attachmentCount` specified in `VkRenderPassCreateInfo` the render pass specified by `VkRenderPassBeginInfo::renderPass` was created with

Valid Usage (Implicit)

- VUID-VkAttachmentSampleLocationsEXT-sampleLocationsInfo-parameter
`sampleLocationsInfo` **must** be a valid `VkSampleLocationsInfoEXT` structure

The `VkSubpassSampleLocationsEXT` structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkSubpassSampleLocationsEXT {
    uint32_t          subpassIndex;
    VkSampleLocationsInfoEXT sampleLocationsInfo;
} VkSubpassSampleLocationsEXT;
```

- `subpassIndex` is the index of the subpass for which the sample locations state is provided.
- `sampleLocationsInfo` is the sample locations state to use for the layout transition of the depth/stencil attachment away from the image layout the attachment is used with in the subpass specified in `subpassIndex`.

If the image referenced by the depth/stencil attachment used in the subpass identified by `subpassIndex` was not created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` or if the subpass does not use a depth/stencil attachment, and `VkPhysicalDeviceSampleLocationsPropertiesEXT::variableSampleLocations` is `VK_TRUE` then the values specified in `sampleLocationsInfo` are ignored.

Valid Usage

- VUID-VkSubpassSampleLocationsEXT-subpassIndex-01532
`subpassIndex` **must** be less than the `subpassCount` specified in `VkRenderPassCreateInfo` the render pass specified by `VkRenderPassBeginInfo::renderPass` was created with

Valid Usage (Implicit)

- VUID-VkSubpassSampleLocationsEXT-sampleLocationsInfo-parameter
`sampleLocationsInfo` **must** be a valid `VkSampleLocationsInfoEXT` structure

The `VkSubpassBeginInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkSubpassBeginInfo {
    VkStructureType sType;
    const void*     pNext;
    VkSubpassContents contents;
} VkSubpassBeginInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `contents` is a `VkSubpassContents` value specifying how the commands in the next subpass will be provided.

Valid Usage (Implicit)

- VUID-VkSubpassBeginInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SUBPASS_BEGIN_INFO`
- VUID-VkSubpassBeginInfo-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkSubpassBeginInfo-contents-parameter
`contents` **must** be a valid `VkSubpassContents` value

Possible values of `vkCmdBeginRenderPass::contents`, specifying how the commands in the first subpass will be provided, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSubpassContents {
    VK_SUBPASS_CONTENTS_INLINE = 0,
    VK_SUBPASS_CONTENTS_SECONDARY_COMMAND_BUFFERS = 1,
} VkSubpassContents;
```

- `VK_SUBPASS_CONTENTS_INLINE` specifies that the contents of the subpass will be recorded inline in the primary command buffer, and secondary command buffers **must** not be executed within the subpass.
- `VK_SUBPASS_CONTENTS_SECONDARY_COMMAND_BUFFERS` specifies that the contents are recorded in secondary command buffers that will be called from the primary command buffer, and `vkCmdExecuteCommands` is the only valid command in the command buffer until `vkCmdNextSubpass` or `vkCmdEndRenderPass`.

If the `pNext` chain of `VkRenderPassBeginInfo` includes a `VkDeviceGroupRenderPassBeginInfo` structure, then that structure includes a device mask and set of render areas for the render pass instance.

The `VkDeviceGroupRenderPassBeginInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkDeviceGroupRenderPassBeginInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           deviceMask;
    uint32_t           deviceRenderAreaCount;
    const VkRect2D*    pDeviceRenderAreas;
} VkDeviceGroupRenderPassBeginInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `deviceMask` is the device mask for the render pass instance.
- `deviceRenderAreaCount` is the number of elements in the `pDeviceRenderAreas` array.

- `pDeviceRenderAreas` is a pointer to an array of `VkRect2D` structures defining the render area for each physical device.

The `deviceMask` serves several purposes. It is an upper bound on the set of physical devices that **can** be used during the render pass instance, and the initial device mask when the render pass instance begins. In addition, commands transitioning to the next subpass in a render pass instance and commands ending the render pass instance, and, accordingly render pass `load`, `store`, and `multisample resolve` operations and subpass dependencies corresponding to the render pass instance, are executed on the physical devices included in the device mask provided here.

If `deviceRenderAreaCount` is not zero, then the elements of `pDeviceRenderAreas` override the value of `VkRenderPassBeginInfo::renderArea`, and provide a render area specific to each physical device. These render areas serve the same purpose as `VkRenderPassBeginInfo::renderArea`, including controlling the region of attachments that are cleared by `VK_ATTACHMENT_LOAD_OP_CLEAR` and that are resolved into resolve attachments.

If this structure is not present, the render pass instance's device mask is the value of `VkDeviceGroupCommandBufferBeginInfo::deviceMask`. If this structure is not present or if `deviceRenderAreaCount` is zero, `VkRenderPassBeginInfo::renderArea` is used for all physical devices.

Valid Usage

- VUID-VkDeviceGroupRenderPassBeginInfo-deviceMask-00905
`deviceMask` **must** be a valid device mask value
- VUID-VkDeviceGroupRenderPassBeginInfo-deviceMask-00906
`deviceMask` **must** not be zero
- VUID-VkDeviceGroupRenderPassBeginInfo-deviceMask-00907
`deviceMask` **must** be a subset of the command buffer's initial device mask
- VUID-VkDeviceGroupRenderPassBeginInfo-deviceRenderAreaCount-00908
`deviceRenderAreaCount` **must** either be zero or equal to the number of physical devices in the logical device
- VUID-VkDeviceGroupRenderPassBeginInfo-offset-06166
The `offset.x` member of any element of `pDeviceRenderAreas` **must** be greater than or equal to 0
- VUID-VkDeviceGroupRenderPassBeginInfo-offset-06167
The `offset.y` member of any element of `pDeviceRenderAreas` **must** be greater than or equal to 0
- VUID-VkDeviceGroupRenderPassBeginInfo-offset-06168
The sum of the `offset.x` and `extent.width` members of any element of `pDeviceRenderAreas` **must** be less than or equal to `maxFramebufferWidth`
- VUID-VkDeviceGroupRenderPassBeginInfo-offset-06169
The sum of the `offset.y` and `extent.height` members of any element of `pDeviceRenderAreas` **must** be less than or equal to `maxFramebufferHeight`
- VUID-VkDeviceGroupRenderPassBeginInfo-extent-08998
The `extent.width` member of any element of `pDeviceRenderAreas` **must** be greater than 0

- VUID-VkDeviceGroupRenderPassBeginInfo-extent-08999
The `extent.height` member of any element of `pDeviceRenderAreas` **must** be greater than 0

Valid Usage (Implicit)

- VUID-VkDeviceGroupRenderPassBeginInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_GROUP_RENDER_PASS_BEGIN_INFO`
- VUID-VkDeviceGroupRenderPassBeginInfo-pDeviceRenderAreas-parameter
If `deviceRenderAreaCount` is not 0, `pDeviceRenderAreas` **must** be a valid pointer to an array of `deviceRenderAreaCount` `VkRect2D` structures

The `VkRenderPassAttachmentBeginInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkRenderPassAttachmentBeginInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           attachmentCount;
    const VkImageView* pAttachments;
} VkRenderPassAttachmentBeginInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `attachmentCount` is the number of attachments.
- `pAttachments` is a pointer to an array of `VkImageView` handles, each of which will be used as the corresponding attachment in the render pass instance.

Valid Usage

- VUID-VkRenderPassAttachmentBeginInfo-pAttachments-03218
Each element of `pAttachments` **must** only specify a single mip level
- VUID-VkRenderPassAttachmentBeginInfo-pAttachments-03219
Each element of `pAttachments` **must** have been created with the `identity swizzle`
- VUID-VkRenderPassAttachmentBeginInfo-pAttachments-04114
Each element of `pAttachments` **must** have been created with `VkImageViewCreateInfo::viewType` not equal to `VK_IMAGE_VIEW_TYPE_3D`

Valid Usage (Implicit)

- VUID-VkRenderPassAttachmentBeginInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_ATTACHMENT_BEGIN_INFO`

- VUID-VkRenderPassAttachmentBeginInfo-pAttachments-parameter
If `attachmentCount` is not 0, `pAttachments` **must** be a valid pointer to an array of `attachmentCount` valid `VkImageView` handles

To query the render area granularity, call:

```
// Provided by VK_VERSION_1_0
void vkGetRenderAreaGranularity(
    VkDevice          device,
    VkRenderPass      renderPass,
    VkExtent2D*       pGranularity);
```

- `device` is the logical device that owns the render pass.
- `renderPass` is a handle to a render pass.
- `pGranularity` is a pointer to a `VkExtent2D` structure in which the granularity is returned.

The conditions leading to an optimal `renderArea` are:

- the `offset.x` member in `renderArea` is a multiple of the `width` member of the returned `VkExtent2D` (the horizontal granularity).
- the `offset.y` member in `renderArea` is a multiple of the `height` member of the returned `VkExtent2D` (the vertical granularity).
- either the `extent.width` member in `renderArea` is a multiple of the horizontal granularity or `offset.x+extent.width` is equal to the `width` of the `framebuffer` in the `VkRenderPassBeginInfo`.
- either the `extent.height` member in `renderArea` is a multiple of the vertical granularity or `offset.y+extent.height` is equal to the `height` of the `framebuffer` in the `VkRenderPassBeginInfo`.

Subpass dependencies are not affected by the render area, and apply to the entire image subresources attached to the framebuffer as specified in the description of [automatic layout transitions](#). Similarly, pipeline barriers are valid even if their effect extends outside the render area.

Valid Usage (Implicit)

- VUID-vkGetRenderAreaGranularity-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetRenderAreaGranularity-renderPass-parameter
`renderPass` **must** be a valid `VkRenderPass` handle
- VUID-vkGetRenderAreaGranularity-pGranularity-parameter
`pGranularity` **must** be a valid pointer to a `VkExtent2D` structure
- VUID-vkGetRenderAreaGranularity-renderPass-parent
`renderPass` **must** have been created, allocated, or retrieved from `device`

To transition to the next subpass in the render pass instance after recording the commands for a subpass, call:

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
void vkCmdNextSubpass(
    VkCommandBuffer                commandBuffer,
    VkSubpassContents              contents);
```

- `commandBuffer` is the command buffer in which to record the command.
- `contents` specifies how the commands in the next subpass will be provided, in the same fashion as the corresponding parameter of [vkCmdBeginRenderPass](#).

The subpass index for a render pass begins at zero when `vkCmdBeginRenderPass` is recorded, and increments each time `vkCmdNextSubpass` is recorded.

After transitioning to the next subpass, the application **can** record the commands for that subpass.

Valid Usage

- VUID-vkCmdNextSubpass-None-00909

The current subpass index **must** be less than the number of subpasses in the render pass minus one

Valid Usage (Implicit)

- VUID-vkCmdNextSubpass-commandBuffer-parameter
`commandBuffer` **must** be a valid [VkCommandBuffer](#) handle
- VUID-vkCmdNextSubpass-contents-parameter
`contents` **must** be a valid [VkSubpassContents](#) value
- VUID-vkCmdNextSubpass-commandBuffer-recording
`commandBuffer` **must** be in the [recording state](#)
- VUID-vkCmdNextSubpass-commandBuffer-cmdpool
The [VkCommandPool](#) that `commandBuffer` was allocated from **must** support [VK_QUEUE_GRAPHICS_BIT](#) operations
- VUID-vkCmdNextSubpass-renderpass
This command **must** only be called inside of a render pass instance
- VUID-vkCmdNextSubpass-bufferlevel
`commandBuffer` **must** be a primary [VkCommandBuffer](#)

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary	Inside	VK_QUEUE_GRAPHICS_BIT	Action State Synchronization

To transition to the next subpass in the render pass instance after recording the commands for a subpass, call:

```
// Provided by VK_VERSION_1_2
void vkCmdNextSubpass2(
    VkCommandBuffer                commandBuffer,
    const VkSubpassBeginInfo*      pSubpassBeginInfo,
    const VkSubpassEndInfo*        pSubpassEndInfo);
```

- `commandBuffer` is the command buffer in which to record the command.
- `pSubpassBeginInfo` is a pointer to a `VkSubpassBeginInfo` structure containing information about the subpass which is about to begin rendering.
- `pSubpassEndInfo` is a pointer to a `VkSubpassEndInfo` structure containing information about how the previous subpass will be ended.

`vkCmdNextSubpass2` is semantically identical to `vkCmdNextSubpass`, except that it is extensible, and that `contents` is provided as part of an extensible structure instead of as a flat parameter.

Valid Usage

- VUID-vkCmdNextSubpass2-None-03102
The current subpass index **must** be less than the number of subpasses in the render pass minus one

Valid Usage (Implicit)

- VUID-vkCmdNextSubpass2-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle

- VUID-vkCmdNextSubpass2-pSubpassBeginInfo-parameter
`pSubpassBeginInfo` **must** be a valid pointer to a valid `VkSubpassBeginInfo` structure
- VUID-vkCmdNextSubpass2-pSubpassEndInfo-parameter
`pSubpassEndInfo` **must** be a valid pointer to a valid `VkSubpassEndInfo` structure
- VUID-vkCmdNextSubpass2-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdNextSubpass2-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdNextSubpass2-renderpass
This command **must** only be called inside of a render pass instance
- VUID-vkCmdNextSubpass2-bufferlevel
`commandBuffer` **must** be a primary `VkCommandBuffer`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary	Inside	VK_QUEUE_GRAPHICS_BIT	Action State Synchronization

To record a command to end a render pass instance after recording the commands for the last subpass, call:

WARNING

This functionality is superseded by [Vulkan Version 1.2](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
void vkCmdEndRenderPass(
    VkCommandBuffer                commandBuffer);
```

- `commandBuffer` is the command buffer in which to end the current render pass instance.

Ending a render pass instance performs any multisample resolve operations on the final subpass.

Valid Usage

- VUID-vkCmdEndRenderPass-None-00910
The current subpass index **must** be equal to the number of subpasses in the render pass minus one
- VUID-vkCmdEndRenderPass-None-07004
If `vkCmdBeginQuery*` was called within a subpass of the render pass, the corresponding `vkCmdEndQuery*` **must** have been called subsequently within the same subpass

Valid Usage (Implicit)

- VUID-vkCmdEndRenderPass-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdEndRenderPass-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdEndRenderPass-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdEndRenderPass-renderpass
This command **must** only be called inside of a render pass instance
- VUID-vkCmdEndRenderPass-bufferlevel
`commandBuffer` **must** be a primary `VkCommandBuffer`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary	Inside	VK_QUEUE_GRAPHICS_BIT	Action State Synchronization

To record a command to end a render pass instance after recording the commands for the last subpass, call:

```
// Provided by VK_VERSION_1_2
void vkCmdEndRenderPass2(
    VkCommandBuffer                      commandBuffer,
    const VkSubpassEndInfo*             pSubpassEndInfo);
```

- **commandBuffer** is the command buffer in which to end the current render pass instance.
- **pSubpassEndInfo** is a pointer to a [VkSubpassEndInfo](#) structure containing information about how the last subpass will be ended.

vkCmdEndRenderPass2 is semantically identical to [vkCmdEndRenderPass](#), except that it is extensible.

Valid Usage

- VUID-vkCmdEndRenderPass2-None-03103
The current subpass index **must** be equal to the number of subpasses in the render pass minus one
- VUID-vkCmdEndRenderPass2-None-07005
If **vkCmdBeginQuery*** was called within a subpass of the render pass, the corresponding **vkCmdEndQuery*** **must** have been called subsequently within the same subpass

Valid Usage (Implicit)

- VUID-vkCmdEndRenderPass2-commandBuffer-parameter
commandBuffer **must** be a valid [VkCommandBuffer](#) handle
- VUID-vkCmdEndRenderPass2-pSubpassEndInfo-parameter
pSubpassEndInfo **must** be a valid pointer to a valid [VkSubpassEndInfo](#) structure
- VUID-vkCmdEndRenderPass2-commandBuffer-recording
commandBuffer **must** be in the [recording state](#)
- VUID-vkCmdEndRenderPass2-commandBuffer-cmdpool
The [VkCommandPool](#) that **commandBuffer** was allocated from **must** support [VK_QUEUE_GRAPHICS_BIT](#) operations
- VUID-vkCmdEndRenderPass2-renderpass
This command **must** only be called inside of a render pass instance
- VUID-vkCmdEndRenderPass2-bufferlevel
commandBuffer **must** be a primary [VkCommandBuffer](#)

Host Synchronization

- Host access to **commandBuffer** **must** be externally synchronized
- Host access to the [VkCommandPool](#) that **commandBuffer** was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary	Inside	VK_QUEUE_GRAPHICS_BIT	Action State Synchronization

The `VkSubpassEndInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkSubpassEndInfo {
    VkStructureType    sType;
    const void*        pNext;
} VkSubpassEndInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

Valid Usage (Implicit)

- VUID-VkSubpassEndInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SUBPASS_END_INFO`
- VUID-VkSubpassEndInfo-pNext-pNext
`pNext` **must** be `NULL`

8.8. Common Render Pass Data Races (Informative)

Due to the complexity of how rendering is performed, there are several ways an application can accidentally introduce a data race, usually by doing something that may seem benign but actually cannot be supported. This section indicates a number of the more common cases as guidelines to help avoid them.

8.8.1. Sampling From a Read-Only Attachment

Vulkan includes read-only layouts for depth/stencil images, that allow the images to be both read during a render pass for the purposes of depth/stencil tests, and read as a non-attachment.

However, because `VK_ATTACHMENT_STORE_OP_STORE` and `VK_ATTACHMENT_STORE_OP_DONT_CARE` may perform write operations, even if no recorded command writes to an attachment, reading from an image while also using it as an attachment with these store operations can result in a data race. If the reads from the non-attachment are performed in a fragment shader where the accessed samples match those covered by the fragment shader, no data race will occur as store operations

are guaranteed to operate after fragment shader execution for the set of samples the fragment covers. Notably, input attachments can also be used for this case. Reading other samples or in any other shader stage can result in unexpected behavior due to the potential for a data race, and validation errors should be generated for doing so. In practice, many applications have shipped reading samples outside of the covered fragment without any observable issue, but there is no guarantee that this will always work, and it is not advisable to rely on this in new or re-worked code bases.

8.8.2. Non-Overlapping Access Between Resources

When relying on non-overlapping accesses between attachments and other resources, it is important to note that [load](#) and [store](#) operations have fairly wide alignment requirements - potentially affecting entire subresources and adjacent depth/stencil aspects. This makes it invalid to access a non-attachment subresource that is simultaneously being used as an attachment where either access performs a write operation.

8.8.3. Depth/Stencil and Input Attachments

When rendering to only the depth OR stencil aspect of an image, an input attachment accessing the other aspect will always result in a data race.

8.8.4. Synchronization Options

There are several synchronization options available to synchronize between accesses to resources within a render pass. Some of the options are outlined below:

- A [VkSubpassDependency](#) in a render pass object can synchronize attachment writes and [multisample resolve operations](#) from a prior subpass for subsequent input attachment reads.
- A [vkCmdPipelineBarrier](#) inside a subpass can synchronize prior attachment writes in the subpass with subsequent input attachment reads.
- If a subresource is used as two separate non-attachment resources, writes to a pixel or individual sample in a fragment shader can be synchronized with access to the same pixel or sample in another fragment shader by using one of the [fragment interlock](#) execution modes.

Chapter 9. Shaders

A shader specifies programmable operations that execute for each vertex, control point, tessellated vertex, primitive, fragment, or workgroup in the corresponding stage(s) of the graphics and compute pipelines.

Graphics pipelines include vertex shader execution as a result of [primitive assembly](#), followed, if enabled, by tessellation control and evaluation shaders operating on [patches](#), geometry shaders, if enabled, operating on primitives, and fragment shaders, if present, operating on fragments generated by [Rasterization](#). In this specification, vertex, tessellation control, tessellation evaluation and geometry shaders are collectively referred to as [pre-rasterization shader stages](#) and occur in the logical pipeline before rasterization. The fragment shader occurs logically after rasterization.

Only the compute shader stage is included in a compute pipeline. Compute shaders operate on compute invocations in a workgroup.

Shaders **can** read from input variables, and read from and write to output variables. Input and output variables **can** be used to transfer data between shader stages, or to allow the shader to interact with values that exist in the execution environment. Similarly, the execution environment provides constants describing capabilities.

Shader variables are associated with execution environment-provided inputs and outputs using *built-in* decorations in the shader. The available decorations for each stage are documented in the following subsections.

9.1. Shader Modules

Shader modules contain *shader code* and one or more entry points. Shaders are selected from a shader module by specifying an entry point as part of [pipeline](#) creation. The stages of a pipeline **can** use shaders that come from different modules. The shader code defining a shader module **must** be in the SPIR-V format, as described by the [Vulkan Environment for SPIR-V](#) appendix.

Shader modules are represented by `VkShaderModule` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkShaderModule)
```

Shader modules are not used in Vulkan SC, but the type has been retained for compatibility [\[SCID-8\]](#).

In Vulkan SC, the shader modules and pipeline state are supplied to an offline compiler which creates a pipeline cache entry which is loaded at [pipeline](#) creation time.

Accordingly, SPIR-V module validation rules defined in the [Vulkan Environment for SPIR-V](#) section are only indirectly applied in Vulkan SC, as the SPIR-V modules that **must** adhere to these validation rules are the inputs of the [offline pipeline cache compilation](#) process, while the Vulkan SC API only consumes the implementation-specific pipeline caches resulting from this process. However, violations of SPIR-V module validation rules **may** be detected and reported by validation

layers when the corresponding pipeline cache entry contains [stage validation index](#) entries with SPIR-V module data.

For completeness, this specification also includes the following shader module creation valid usage clauses that are expected to be referred to by validation layers to report SPIR-V module validation errors:

- Let **pCode** be a pointer to the SPIR-V module data of the [pipeline cache stage validation index entry](#).
- Let **codeSize** be the corresponding member of the [VkPipelineCacheStageValidationIndexEntry](#) structure describing the [pipeline cache stage validation index entry](#).

Valid Usage

- VUID-VkShaderModuleCreateInfo-codeSize-08735
If **pCode** is a pointer to SPIR-V code, **codeSize** **must** be a multiple of 4
- VUID-VkShaderModuleCreateInfo-pCode-08736
If **pCode** is a pointer to SPIR-V code, **pCode** **must** point to valid SPIR-V code, formatted and packed as described by the [Khronos SPIR-V Specification](#)
- VUID-VkShaderModuleCreateInfo-pCode-08737
If **pCode** is a pointer to SPIR-V code, **pCode** **must** adhere to the validation rules described by the [Validation Rules within a Module](#) section of the [SPIR-V Environment](#) appendix
- VUID-VkShaderModuleCreateInfo-pCode-08738
If **pCode** is a pointer to SPIR-V code, **pCode** **must** declare the **Shader** capability for SPIR-V code
- VUID-VkShaderModuleCreateInfo-pCode-08739
If **pCode** is a pointer to SPIR-V code, **pCode** **must** not declare any capability that is not supported by the API, as described by the [Capabilities](#) section of the [SPIR-V Environment](#) appendix
- VUID-VkShaderModuleCreateInfo-pCode-08740
If **pCode** is a pointer to SPIR-V code, and **pCode** declares any of the capabilities listed in the [SPIR-V Environment](#) appendix, one of the corresponding requirements **must** be satisfied
- VUID-VkShaderModuleCreateInfo-pCode-08741
If **pCode** is a pointer to SPIR-V code, **pCode** **must** not declare any SPIR-V extension that is not supported by the API, as described by the [Extension](#) section of the [SPIR-V Environment](#) appendix
- VUID-VkShaderModuleCreateInfo-pCode-08742
If **pCode** is a pointer to SPIR-V code, and **pCode** declares any of the SPIR-V extensions listed in the [SPIR-V Environment](#) appendix, one of the corresponding requirements **must** be satisfied

9.2. Binding Shaders

Before a shader can be used it **must** be first bound to the command buffer.

Calling `vkCmdBindPipeline` binds all stages corresponding to the `VkPipelineBindPoint`.

The following table describes the relationship between shader stages and pipeline bind points:

Shader stage	Pipeline bind point	behavior controlled
• <code>VK_SHADER_STAGE_VERTEX_BIT</code> • <code>VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT</code> • <code>VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT</code> • <code>VK_SHADER_STAGE_GEOMETRY_BIT</code> • <code>VK_SHADER_STAGE_FRAGMENT_BIT</code>	<code>VK_PIPELINE_BIND_POINT_GRAPHICS</code>	all <code>drawing commands</code>
• <code>VK_SHADER_STAGE_COMPUTE_BIT</code>	<code>VK_PIPELINE_BIND_POINT_COMPUTE</code>	all <code>dispatch commands</code>

9.3. Shader Execution

At each stage of the pipeline, multiple invocations of a shader **may** execute simultaneously. Further, invocations of a single shader produced as the result of different commands **may** execute simultaneously. The relative execution order of invocations of the same shader type is undefined. Shader invocations **may** complete in a different order than that in which the primitives they originated from were drawn or dispatched by the application. However, fragment shader outputs are written to attachments in `rasterization order`.

The relative execution order of invocations of different shader types is largely undefined. However, when invoking a shader whose inputs are generated from a previous pipeline stage, the shader invocations from the previous stage are guaranteed to have executed far enough to generate input values for all required inputs.

9.3.1. Shader Termination

A shader invocation that is *terminated* has finished executing instructions.

Executing `OpReturn` in the entry point, or executing `OpTerminateInvocation` in any function will terminate an invocation. Implementations **may** also terminate a shader invocation when `OpKill` is executed in any function; otherwise it becomes a `helper invocation`.

In addition to the above conditions, `helper invocations` **may** be terminated when all non-helper invocations in the same `derivative group` either terminate or become `helper invocations`.

A shader stage for a given command completes execution when all invocations for that stage have

terminated.

NOTE

Depending on the implementation, `OpKill` will be functionally equivalent to either `OpTerminateInvocation` or `OpDemoteToHelperInvocation`. To obtain the most predictable behavior, shader authors should use `OpTerminateInvocation` or `OpDemoteToHelperInvocation` rather than `OpKill` wherever possible.

9.4. Shader Out-of-Bounds Memory Access

Shader accesses to memory are not automatically bounds checked by the implementation. Applications **must** not execute operations that would access out of bounds memory locations unless some form of bounds checking is enabled. An access is considered out of bounds if any part of the access is outside of any specified memory range, whether that is the array length specified in a shader or a range specified in the API (e.g. descriptor size).

NOTE

External tooling such as the Vulkan Validation Layers can be used to help validate that accesses are not out of bounds.

An access **can** be independently out of bounds for each range that applies; if one is bounds checked and the others are not, behavior is still undefined.

For example, given the following shader declaration

NOTE

```
// Buffer type
struct MySSBO {
    uint32_t data[2];
};
```

accessing `data` at an index greater than 1 is undefined behavior, whether the underlying buffer is bigger than that or not.

Vulkan provides functionality that enables automatic bounds checking in some cases, as outlined below.

NOTE

Automatic bounds checking can be used to ensure that accesses outside of certain bounds have predictable results, acting as a safety net for untrusted code, or simply as a way for applications to avoid their own bounds checks. While there may be a performance cost for enabling these features, they should not be slower than an application performing equivalent checks. Automatic checks do not necessarily account for all possible bounds - e.g. [Robust Buffer Access](#) will not prevent undefined behavior in the buffer access example in the prior note.

9.4.1. Robust Buffer Access

Robust buffer access **can** be enabled by the `robustBufferAccess` feature.

When robust buffer access is enabled, access to a buffer via a descriptor is bounds checked against the range specified for the descriptor, and access to vertex input data is bounds checked against the bound vertex buffer range. Reads from a vertex input **may** instead be bounds checked against a range rounded down to the nearest multiple of the stride of its binding.

NOTE

The range of a descriptor is not necessarily equivalent to the size of the underlying resource; applications may suballocate descriptors from larger buffers, for instance. The APIs specifying the descriptor range vary between resource types and descriptor interfaces, but for example include the ranges specified by [VkDescriptorBufferInfo](#) or [VkBufferViewCreateInfo](#).

If any vertex input read is outside of the checked range, all other vertex input reads through the same binding in the same shader invocation **may** behave as if they were outside of the checked range.

If any access to a uniform, storage, uniform texel, or storage texel buffer is outside of the checked range, any access of the same type (write, read-modify-write, or read) to the same buffer that is less than 16 bytes away from the first access **may** behave as if it is also outside of the checked range.

Any non-atomic access to a uniform, storage, uniform texel, or storage texel buffer wider than 32-bits **may** be treated as multiple 32-bit accesses that are separately bounds checked.

Writes to a storage or storage texel buffer outside of the checked range will either be discarded, or modify values within the memory range(s) bound to the underlying buffer (including outside of the checked range). They will not modify any other memory.

NOTE

Non-atomic writes outside of the checked range **can** lead to data races, as the application has no control over where the data will be written.

Atomic read-modify-write operations to a storage or storage texel buffer outside of the checked range will behave the same as a write outside of the checked range, but will return an undefined value.

Reading a uniform, storage, uniform texel, or storage texel buffer outside of the checked range will return one of the following values:

- Values from anywhere within the memory range(s) bound to the underlying buffer object, which **may** include bytes beyond the size of the buffer itself.
- Zero values
- For 4-component vectors, a value of (0,0,0,x), where x is any of
 - 0, 1, or the maximum positive integer value for integer components
 - 0.0 or 1.0 for floating-point components
- The value of the last store to the same out-of-bounds location in the same shader invocation.
 - Using the **Volatile/VolatileTexel** memory/image operand, the **Volatile** memory semantic, or the **Volatile** decoration to load the value will prevent prior stored values from being returned.

NOTE

Getting the value of the previous store is possible as implementations are free to optimize multiple accesses in the general case. There are several ways this **can** be prevented, but using volatile loads is by far the simplest.

Reads from a vertex input outside of the checked range will produce one of the following values:

- Values from anywhere within the memory range(s) bound to the underlying buffer object, which **may** include bytes beyond the size of the buffer itself, converted via [input extraction](#).
- Zero values, converted via [input extraction](#).
- Zero values
- For 4-component vectors, a value of (0,0,0,x), where x is any of
 - 0, 1, or the maximum positive integer value for integer components
 - 0.0 or 1.0 for floating-point components

9.4.2. Robust Buffer Access 2

Robust buffer access 2 **can** be enabled by the `robustBufferAccess2` feature.

When robust buffer access 2 is enabled, access to a buffer via a descriptor is bounds checked against the range specified for the descriptor, and access to vertex input data is bounds checked against the bound vertex buffer range, similarly to [Robust Buffer Access](#), but with tighter bounds on the results.

Accesses to a uniform buffer **may** instead be bounds checked against a range rounded up to `robustUniformBufferAccessSizeAlignment`. Accesses inside the aligned range **may** behave as if they are in bounds, even if they are outside of the unaligned descriptor range, and access memory accordingly. The same is true for accesses to a storage buffer, using the `robustStorageBufferAccessSizeAlignment` limit instead.

NOTE

To avoid unexpected data races between neighboring descriptor ranges, applications may wish to ensure suballocated ranges of buffers are aligned to these limits.

Any access to a uniform, storage, uniform texel, or storage texel buffer wider than 32-bits **may** be treated as multiple 32-bit accesses that are separately bounds checked.

NOTE

Accesses to null descriptors are not considered out-of-bounds and have separate behavior controlled by the `nullDescriptor` feature.

Writes to a storage or storage texel buffer outside of the checked range will not modify any memory.

Atomic read-modify-write operations to a storage or storage texel buffer outside of the checked range will behave the same as a write outside of the checked range, but will return an undefined value.

Reads from a uniform or storage buffer outside of the checked range will return zero values. If a

value was previously written to the same out of bounds location in the same shader invocation, that value **may** be returned instead; using the `Volatile/VolatileTexel` memory/image operand, the `Volatile` memory semantic, or the `Volatile` decoration to load the value will prevent prior stored values from being returned.

Reading a uniform texel or storage texel buffer outside of the checked range will produce zero values, but `component substitution` will still be applied based on the buffer view's format, with the resulting value returned to the shader. If a value was previously written to the same out of bounds location in the same shader invocation, that value **may** be returned instead; using the `Volatile/VolatileTexel` memory/image operand, the `Volatile` memory semantic, or the `Volatile` decoration to load the value will prevent prior stored values from being returned.

Reads from a vertex input outside of the checked range will produce zero values, but `input extraction` will still be applied, filling missing G, B, or A components with (0,0,1).

9.4.3. Image Sampling

Sampling operations on an image descriptor are always well-defined when coordinates exceeding the dimensions specified for the descriptor are accessed, as described in the [Wrapping Operation](#) section.

9.4.4. Robust Image Access

Robust image access **can** be enabled by the `robustImageAccess` feature.

If robust image access is enabled, accesses to image descriptors are bounds checked against the image view dimensions specified for the descriptor.

Writes or atomic read-modify-write operations to a storage image outside of the checked dimensions will not modify any memory.

Reads, atomic read-modify-write operations, or fetches from images outside of the checked dimensions will return zero values, with (0,0,1) or [eq]#(0,0,0) values [inserted for missing G, B, or A components](#) based on the format.

If a value was previously written to the same out of bounds location in the same shader invocation, that value **may** be returned instead; using the `VolatileTexel` image operand, the `Volatile` memory semantic, or the `Volatile` decoration to load the value will prevent prior stored values from being returned.

9.4.5. Robust Image Access 2

NOTE

This is largely identical to [Robust Image Access](#); the only difference being that the alpha channel must be replaced with 1, rather than 1 or 0, for out of bounds texel access.

Robust image access 2 **can** be enabled by the `robustImageAccess2` feature.

If robust image access 2 is enabled, accesses to image descriptors are bounds checked against the

image view dimensions specified for the descriptor.

Writes or atomic read-modify-write operations to a storage image outside of the checked dimensions will not modify any memory.

Reads, atomic read-modify-write operations, or fetches from images outside of the checked dimensions will return zero values, with (0,0,1) values [inserted for missing G, B, or A components](#) based on the format.

If a value was previously written to the same out of bounds location in the same shader invocation, that value **may** be returned instead; using the `VolatileTexel` image operand, the `Volatile` memory semantic, or the `Volatile` decoration to load the value will prevent prior stored values from being returned.

9.5. Shader Memory Access Ordering

The order in which image or buffer memory is read or written by shaders is largely undefined. For some shader types (vertex, tessellation evaluation, and in some cases, fragment), even the number of shader invocations that **may** perform loads and stores is undefined.

In particular, the following rules apply:

- [Vertex](#) and [tessellation evaluation](#) shaders will be invoked at least once for each unique vertex, as defined in those sections.
- [Fragment](#) shaders will be invoked zero or more times, as defined in that section.
- The relative execution order of invocations of the same shader type is undefined. A store issued by a shader when working on primitive B might complete prior to a store for primitive A, even if primitive A is specified prior to primitive B. This applies even to fragment shaders; while fragment shader outputs are always written to the framebuffer in [rasterization order](#), stores executed by fragment shader invocations are not.
- The relative execution order of invocations of different shader types is largely undefined.

NOTE

The above limitations on shader invocation order make some forms of synchronization between shader invocations within a single set of primitives unimplementable. For example, having one invocation poll memory written by another invocation assumes that the other invocation has been launched and will complete its writes in finite time.

The [Memory Model](#) appendix defines the terminology and rules for how to correctly communicate between shader invocations, such as when a write is [Visible-To](#) a read, and what constitutes a [Data Race](#). Applications **must** not cause a data race.

9.6. Shader Inputs and Outputs

Data is passed into and out of shaders using variables with input or output storage class, respectively. User-defined inputs and outputs are connected between stages by matching their `Location` decorations. Additionally, data **can** be provided by or communicated to special functions

provided by the execution environment using **BuiltIn** decorations.

In many cases, the same **BuiltIn** decoration **can** be used in multiple shader stages with similar meaning. The specific behavior of variables decorated as **BuiltIn** is documented in the following sections.

9.7. Vertex Shaders

Each vertex shader invocation operates on one vertex and its associated **vertex attribute** data, and outputs one vertex and associated data. Graphics pipelines **must** include a vertex shader, and the vertex shader stage is always the first shader stage in the graphics pipeline.

9.7.1. Vertex Shader Execution

A vertex shader **must** be executed at least once for each vertex specified by a drawing command. If the subpass includes multiple views in its view mask, the shader **may** be invoked separately for each view. During execution, the shader is presented with the index of the vertex and instance for which it has been invoked. Input variables declared in the vertex shader are filled by the implementation with the values of vertex attributes associated with the invocation being executed.

If the same vertex is specified multiple times in a drawing command (e.g. by including the same index value multiple times in an index buffer) the implementation **may** reuse the results of vertex shading if it can statically determine that the vertex shader invocations will produce identical results.

NOTE

It is implementation-dependent when and if results of vertex shading are reused, and thus how many times the vertex shader will be executed. This is true also if the vertex shader contains stores or atomic operations (see **vertexPipelineStoresAndAtomics**).

9.8. Tessellation Control Shaders

The tessellation control shader is used to read an input patch provided by the application and to produce an output patch. Each tessellation control shader invocation operates on an input patch (after all control points in the patch are processed by a vertex shader) and its associated data, and outputs a single control point of the output patch and its associated data, and **can** also output additional per-patch data. The input patch is sized according to the **patchControlPoints** member of **VkPipelineTessellationStateCreateInfo**, as part of input assembly.

The input patch can also be dynamically sized with **patchControlPoints** parameter of **vkCmdSetPatchControlPointsEXT**.

To **dynamically set** the number of control points per patch, call:

```
// Provided by VK_EXT_extended_dynamic_state2
void vkCmdSetPatchControlPointsEXT(
    VkCommandBuffer          commandBuffer,
    uint32_t                 patchControlPoints);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `patchControlPoints` specifies the number of control points per patch.

This command sets the number of control points per patch for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineTessellationStateCreateInfo::patchControlPoints` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetPatchControlPointsEXT-None-09422
At least one of the following **must** be true:
 - The `extendedDynamicState2PatchControlPoints` feature is enabled
- VUID-vkCmdSetPatchControlPointsEXT-patchControlPoints-04874
`patchControlPoints` **must** be greater than zero and less than or equal to `VkPhysicalDeviceLimits::maxTessellationPatchSize`

Valid Usage (Implicit)

- VUID-vkCmdSetPatchControlPointsEXT-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetPatchControlPointsEXT-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdSetPatchControlPointsEXT-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHIC S_BIT	State

The size of the output patch is controlled by the `OpExecutionMode OutputVertices` specified in the tessellation control or tessellation evaluation shaders, which **must** be specified in at least one of the shaders. The size of the input and output patches **must** each be greater than zero and less than or equal to `VkPhysicalDeviceLimits::maxTessellationPatchSize`.

9.8.1. Tessellation Control Shader Execution

A tessellation control shader is invoked at least once for each *output* vertex in a patch. If the subpass includes multiple views in its view mask, the shader **may** be invoked separately for each view.

Inputs to the tessellation control shader are generated by the vertex shader. Each invocation of the tessellation control shader **can** read the attributes of any incoming vertices and their associated data. The invocations corresponding to a given patch execute logically in parallel, with undefined relative execution order. However, the `OpControlBarrier` instruction **can** be used to provide limited control of the execution order by synchronizing invocations within a patch, effectively dividing tessellation control shader execution into a set of phases. Tessellation control shaders will read undefined values if one invocation reads a per-vertex or per-patch output written by another invocation at any point during the same phase, or if two invocations attempt to write different values to the same per-patch output in a single phase.

9.9. Tessellation Evaluation Shaders

The Tessellation Evaluation Shader operates on an input patch of control points and their associated data, and a single input barycentric coordinate indicating the invocation's relative position within the subdivided patch, and outputs a single vertex and its associated data.

9.9.1. Tessellation Evaluation Shader Execution

A tessellation evaluation shader is invoked at least once for each unique vertex generated by the tessellator. If the subpass includes multiple views in its view mask, the shader **may** be invoked separately for each view.

9.10. Geometry Shaders

The geometry shader operates on a group of vertices and their associated data assembled from a single input primitive, and emits zero or more output primitives and the group of vertices and their associated data required for each output primitive.

9.10.1. Geometry Shader Execution

A geometry shader is invoked at least once for each primitive produced by the tessellation stages, or at least once for each primitive generated by [primitive assembly](#) when tessellation is not in use. A shader can request that the geometry shader runs multiple [instances](#). A geometry shader is invoked at least once for each instance. If the subpass includes multiple views in its view mask, the shader **may** be invoked separately for each view.

9.11. Fragment Shaders

Fragment shaders are invoked as a [fragment operation](#) in a graphics pipeline. Each fragment shader invocation operates on a single fragment and its associated data. With few exceptions, fragment shaders do not have access to any data associated with other fragments and are considered to execute in isolation of fragment shader invocations associated with other fragments.

9.12. Compute Shaders

Compute shaders are invoked via [dispatching commands](#). In general, they have access to similar resources as shader stages executing as part of a graphics pipeline.

Compute workloads are formed from groups of work items called workgroups and processed by the compute shader in the current compute pipeline. A workgroup is a collection of shader invocations that execute the same shader, potentially in parallel. Compute shaders execute in *global workgroups* which are divided into a number of *local workgroups* with a size that **can** be set by assigning a value to the [LocalSize](#) execution mode or via an object decorated by the [WorkgroupSize](#) decoration. An invocation within a local workgroup **can** share data with other members of the local workgroup through shared variables and issue memory and control flow barriers to synchronize with other members of the local workgroup.

9.13. Interpolation Decorations

Variables in the [Input](#) storage class in a fragment shader's interface are interpolated from the values specified by the primitive being rasterized.

NOTE

Interpolation decorations can be present on input and output variables in pre-rasterization shaders but have no effect on the interpolation performed.

An undecorated input variable will be interpolated with perspective-correct interpolation according to the primitive type being rasterized. [Lines](#) and [polygons](#) are interpolated in the same way as the primitive's clip coordinates. If the [NoPerspective](#) decoration is present, linear interpolation is instead used for [lines](#) and [polygons](#). For points, as there is only a single vertex, input values are never interpolated and instead take the value written for the single vertex.

If the [Flat](#) decoration is present on an input variable, the value is not interpolated, and instead takes its value directly from the [provoking vertex](#). Fragment shader inputs that are signed or unsigned integers, integer vectors, or any double-precision floating-point type **must** be decorated with [Flat](#).

Interpolation of input variables is performed at an implementation-defined position within the fragment area being shaded. The position is further constrained as follows:

- If the **Centroid** decoration is used, the interpolation position used for the variable **must** also fall within the bounds of the primitive being rasterized.
- If the **Sample** decoration is used, the interpolation position used for the variable **must** be at the position of the sample being shaded by the current fragment shader invocation.
- If a sample count of 1 is used, the interpolation position **must** be at the center of the fragment area.

NOTE

As **Centroid** constrains the interpolation position to lie within the covered area of the primitive, using it may cause the position to differ between neighboring fragments when it otherwise would not. Derivatives calculated based on these differing locations can produce inconsistent results compared to undecorated inputs. Thus using **Centroid** with input variables used in derivative calculations is not recommended.

9.14. Static Use

A SPIR-V module declares a global object in memory using the **OpVariable** instruction, which results in a pointer **x** to that object. A specific entry point in a SPIR-V module is said to *statically use* that object if that entry point's call tree contains a function containing a instruction with **x** as an **id** operand. A shader entry point also *statically uses* any variables explicitly declared in its interface.

9.15. Scope

A *scope* describes a set of shader invocations, where each such set is a *scope instance*. Each invocation belongs to one or more scope instances, but belongs to no more than one scope instance for each scope.

The operations available between invocations in a given scope instance vary, with smaller scopes generally able to perform more operations, and with greater efficiency.

9.15.1. Cross Device

All invocations executed in a Vulkan instance fall into a single *cross device scope instance*.

Whilst the **CrossDevice** scope is defined in SPIR-V, it is disallowed in Vulkan. API **synchronization** commands **can** be used to communicate between devices.

9.15.2. Device

All invocations executed on a single device form a *device scope instance*.

If the **vulkanMemoryModel** and **vulkanMemoryModelDeviceScope** features are enabled, this scope is represented in SPIR-V by the **Device Scope**, which **can** be used as a **Memory Scope** for barrier and atomic operations.

If both the `shaderDeviceClock` and `vulkanMemoryModelDeviceScope` features are enabled, using the `Device Scope` with the `OpReadClockKHR` instruction will read from a clock that is consistent across invocations in the same device scope instance.

There is no method to synchronize the execution of these invocations within SPIR-V, and this **can** only be done with API synchronization primitives.

Invocations executing on different devices in a device group operate in separate device scope instances.

9.15.3. Queue Family

Invocations executed by queues in a given queue family form a *queue family scope instance*.

This scope is identified in SPIR-V as the `QueueFamily Scope` if the `vulkanMemoryModel` feature is enabled, or if not, the `Device Scope`, which **can** be used as a `Memory Scope` for barrier and atomic operations.

If the `shaderDeviceClock` feature is enabled, but the `vulkanMemoryModelDeviceScope` feature is not enabled, using the `Device Scope` with the `OpReadClockKHR` instruction will read from a clock that is consistent across invocations in the same queue family scope instance.

There is no method to synchronize the execution of these invocations within SPIR-V, and this **can** only be done with API synchronization primitives.

Each invocation in a queue family scope instance **must** be in the same [device scope instance](#).

9.15.4. Command

Any shader invocations executed as the result of a single command such as `vkCmdDispatch` or `vkCmdDraw` form a *command scope instance*. For indirect drawing commands with `drawCount` greater than one, invocations from separate draws are in separate command scope instances.

There is no specific `Scope` for communication across invocations in a command scope instance. As this has a clear boundary at the API level, coordination here **can** be performed in the API, rather than in SPIR-V.

Each invocation in a command scope instance **must** be in the same [queue-family scope instance](#).

For shaders without defined [workgroups](#), this set of invocations forms an *invocation group* as defined in the [SPIR-V specification](#).

9.15.5. Primitive

Any fragment shader invocations executed as the result of rasterization of a single primitive form a *primitive scope instance*.

There is no specific `Scope` for communication across invocations in a primitive scope instance.

Any generated [helper invocations](#) are included in this scope instance.

Each invocation in a primitive scope instance **must** be in the same [command scope instance](#).

Any input variables decorated with **Flat** are uniform within a primitive scope instance.

9.15.6. Workgroup

A *local workgroup* is a set of invocations that can synchronize and share data with each other using memory in the **Workgroup** storage class.

The **Workgroup Scope** can be used as both an **Execution Scope** and **Memory Scope** for barrier and atomic operations.

Each invocation in a local workgroup **must** be in the same [command scope instance](#).

Only compute shaders have defined workgroups - other shader types **cannot** use workgroup functionality. For shaders that have defined workgroups, this set of invocations forms an *invocation group* as defined in the [SPIR-V specification](#).

The amount of storage consumed by the variables declared with the **Workgroup** storage class is implementation-dependent. However, the amount of storage consumed may not exceed the largest block size that would be obtained if all active variables declared with **Workgroup** storage class were assigned offsets in an arbitrary order by successively taking the smallest valid offset according to the [Standard Storage Buffer Layout](#) rules, and with **Boolean** values considered as 32-bit integer values for the purpose of this calculation. (This is equivalent to using the GLSL std430 layout rules.)

9.15.7. Subgroup

A *subgroup* (see the subsection “Control Flow” of section 2 of the SPIR-V 1.3 Revision 1 specification) is a set of invocations that can synchronize and share data with each other efficiently.

The **Subgroup Scope** can be used as both an **Execution Scope** and **Memory Scope** for barrier and atomic operations. Other [subgroup features](#) allow the use of [group operations](#) with subgroup scope.

If the **shaderSubgroupClock** feature is enabled, using the **Subgroup Scope** with the **OpReadClockKHR** instruction will read from a clock that is consistent across invocations in the same subgroup.

For [shaders that have defined workgroups](#), each invocation in a subgroup **must** be in the same [local workgroup](#).

In other shader stages, each invocation in a subgroup **must** be in the same [device scope instance](#).

Only [shader stages that support subgroup operations](#) have defined subgroups.

NOTE

Subgroups are not guaranteed to be a subset of a single command in [shaders that do not have defined workgroups](#). Values that are guaranteed to be uniform for a given command or sub command may then not be uniform for the subgroup, and vice versa. As such, applications must take care when dealing with mixed uniformity.

A somewhat common example of this would something like trying to optimize access to per-draw data using subgroup operations:

```

buffer { uint draw_data[]; };

flat in int vDrawID; // Passed through from vertex shader

void main()
{
    uint local_draw_data = subgroupBroadcastFirst(draw_data
[local_draw_data]);
}

```

This can be done in an attempt to optimize the shader to only perform the loads once per subgroup. However, if the implementation packs multiple draws into a single subgroup, invocations from draws with a different drawID are now receiving data from the wrong invocation. Applications should rely on implementations to do this kind of optimization automatically where the implementation can, rather than trying to force it.

9.15.8. Quad

A *quad scope instance* is formed of four shader invocations.

In a fragment shader, each invocation in a quad scope instance is formed of invocations in neighboring framebuffer locations (x_i, y_i) , where:

- i is the index of the invocation within the scope instance.
- w and h are the number of pixels the fragment covers in the x and y axes.
- w and h are identical for all participating invocations.
- $(x_0) = (x_1 - w) = (x_2) = (x_3 - w)$
- $(y_0) = (y_1) = (y_2 - h) = (y_3 - h)$
- Each invocation has the same layer and sample indices.

In all shaders, each invocation in a quad scope instance is formed of invocations in adjacent subgroup invocation indices (s_i) , where:

- i is the index of the invocation within the quad scope instance.
- $(s_0) = (s_1 - 1) = (s_2 - 2) = (s_3 - 3)$
- s_0 is an integer multiple of 4.

Each invocation in a quad scope instance **must** be in the same [subgroup](#).

In a fragment shader, each invocation in a quad scope instance **must** be in the same [primitive scope instance](#).

Fragment and compute shaders have defined quad scope instances. If the [quadOperationsInAllStages](#) limit is supported, any [shader stages that support subgroup operations](#) also have defined quad scope instances.

9.15.9. Fragment Interlock

A *fragment interlock scope instance* is formed of fragment shader invocations based on their framebuffer locations (x,y,layer,sample), executed by commands inside a single [subpass](#).

The specific set of invocations included varies based on the execution mode as follows:

- If the [SampleInterlockOrderedEXT](#) or [SampleInterlockUnorderedEXT](#) execution modes are used, only invocations with identical framebuffer locations (x,y,layer,sample) are included.
- If the [PixelInterlockOrderedEXT](#) or [PixelInterlockUnorderedEXT](#) execution modes are used, fragments with different sample ids are also included.
- If the [ShadingRateInterlockOrderedEXT](#) or [ShadingRateInterlockUnorderedEXT](#) execution modes are used, fragments from neighboring framebuffer locations are also included. The [fragment shading rate](#) determines these fragments.

Only fragment shaders with one of the above execution modes have defined fragment interlock scope instances.

There is no specific [Scope](#) value for communication across invocations in a fragment interlock scope instance. However, this is implicitly used as a memory scope by [OpBeginInvocationInterlockEXT](#) and [OpEndInvocationInterlockEXT](#).

Each invocation in a fragment interlock scope instance **must** be in the same [queue family scope instance](#).

9.15.10. Invocation

The smallest *scope* is a single invocation; this is represented by the [Invocation Scope](#) in SPIR-V.

Fragment shader invocations **must** be in a [primitive scope instance](#).

Invocations in [fragment shaders that have a defined fragment interlock scope](#) **must** be in a [fragment interlock scope instance](#).

Invocations in [shaders that have defined workgroups](#) **must** be in a [local workgroup](#).

Invocations in [shaders that have a defined subgroup scope](#) **must** be in a [subgroup](#).

Invocations in [shaders that have a defined quad scope](#) **must** be in a [quad scope instance](#).

All invocations in all stages **must** be in a [command scope instance](#).

9.16. Group Operations

Group operations are executed by multiple invocations within a [scope instance](#); with each invocation involved in calculating the result. This provides a mechanism for efficient communication between invocations in a particular scope instance.

Group operations all take a [Scope](#) defining the desired [scope instance](#) to operate within. Only the [Subgroup](#) scope **can** be used for these operations; the [subgroupSupportedOperations](#) limit defines

which types of operation **can** be used.

9.16.1. Basic Group Operations

Basic group operations include the use of `OpGroupNonUniformElect`, `OpControlBarrier`, `OpMemoryBarrier`, and atomic operations.

`OpGroupNonUniformElect` **can** be used to choose a single invocation to perform a task for the whole group. Only the invocation with the lowest id in the group will return `true`.

The [Memory Model](#) appendix defines the operation of barriers and atomics.

9.16.2. Vote Group Operations

The vote group operations allow invocations within a group to compare values across a group. The types of votes enabled are:

- Do all active group invocations agree that an expression is true?
- Do any active group invocations evaluate an expression to true?
- Do all active group invocations have the same value of an expression?

NOTE

These operations are useful in combination with control flow in that they allow for developers to check whether conditions match across the group and choose potentially faster code-paths in these cases.

9.16.3. Arithmetic Group Operations

The arithmetic group operations allow invocations to perform scans and reductions across a group. The operators supported are add, mul, min, max, and, or, xor.

For reductions, every invocation in a group will obtain the cumulative result of these operators applied to all values in the group. For exclusive scans, each invocation in a group will obtain the cumulative result of these operators applied to all values in invocations with a lower index in the group. Inclusive scans are identical to exclusive scans, except the cumulative result includes the operator applied to the value in the current invocation.

The order in which these operators are applied is implementation-dependent.

9.16.4. Ballot Group Operations

The ballot group operations allow invocations to perform more complex votes across the group. The ballot functionality allows all invocations within a group to provide a boolean value and get as a result what each invocation provided as their boolean value. The broadcast functionality allows values to be broadcast from an invocation to all other invocations within the group.

9.16.5. Shuffle Group Operations

The shuffle group operations allow invocations to read values from other invocations within a

group.

9.16.6. Shuffle Relative Group Operations

The shuffle relative group operations allow invocations to read values from other invocations within the group relative to the current invocation in the group. The relative operations supported allow data to be shifted up and down through the invocations within a group.

9.16.7. Clustered Group Operations

The clustered group operations allow invocations to perform an operation among partitions of a group, such that the operation is only performed within the group invocations within a partition. The partitions for clustered group operations are consecutive power-of-two size groups of invocations and the cluster size **must** be known at pipeline creation time. The operations supported are add, mul, min, max, and, or, xor.

9.17. Quad Group Operations

Quad group operations (`OpGroupNonUniformQuad*`) are a specialized type of [group operations](#) that only operate on [quad scope instances](#). Whilst these instructions do include a `Scope` parameter, this scope is always overridden; only the [quad scope instance](#) is included in its execution scope.

Fragment shaders that statically execute either `OpGroupNonUniformQuadBroadcast` or `OpGroupNonUniformQuadSwap` **must** launch sufficient invocations to ensure their correct operation; additional [helper invocations](#) are launched for framebuffer locations not covered by rasterized fragments if necessary.

The index used to select participating invocations is `i`, as described for a [quad scope instance](#), defined as the *quad index* in the [SPIR-V specification](#).

For `OpGroupNonUniformQuadBroadcast` this value is equal to `Index`. For `OpGroupNonUniformQuadSwap`, it is equal to the implicit `Index` used by each participating invocation.

9.18. Derivative Operations

Derivative operations calculate the partial derivative for an expression `P` as a function of an invocation's `x` and `y` coordinates.

Derivative operations operate on a set of invocations known as a *derivative group* as defined in the [SPIR-V specification](#).

A derivative group in a fragment shader is equivalent to the [primitive scope instance](#).

Derivatives are calculated assuming that `P` is piecewise linear and continuous within the derivative group.

The following control-flow restrictions apply to derivative operations:

- dynamic instances of explicit derivative instructions (`OpDPdx*`, `OpDPdy*`, and `OpFwidth*`) **must** be

executed in control flow that is uniform within a derivative group.

- dynamic instances of implicit derivative operations **can** be executed in control flow that is not uniform within the derivative group, but results are undefined.

Fragment shaders that statically execute derivative operations **must** launch sufficient invocations to ensure their correct operation; additional [helper invocations](#) are launched for framebuffer locations not covered by rasterized fragments if necessary.

Derivative operations calculate their results as the difference between the result of P across invocations in the quad. For fine derivative operations ([OpDPdxFine](#) and [OpDPdyFine](#)), the values of $DPdx(P_i)$ are calculated as

$$DPdx(P_0) = DPdx(P_1) = P_1 - P_0$$

$$DPdx(P_2) = DPdx(P_3) = P_3 - P_2$$

and the values of $DPdy(P_i)$ are calculated as

$$DPdy(P_0) = DPdy(P_2) = P_2 - P_0$$

$$DPdy(P_1) = DPdy(P_3) = P_3 - P_1$$

where i is the index of each invocation as described in [Quad](#).

Coarse derivative operations ([OpDPdxCoarse](#) and [OpDPdyCoarse](#)), calculate their results in roughly the same manner, but **may** only calculate two values instead of four (one for each of $DPdx$ and $DPdy$), reusing the same result no matter the originating invocation. If an implementation does this, it **should** use the fine derivative calculations described for P_0 .

NOTE

Derivative values are calculated between fragments rather than pixels. If the fragment shader invocations involved in the calculation cover multiple pixels, these operations cover a wider area, resulting in larger derivative values. This in turn will result in a coarser LOD being selected for image sampling operations using derivatives.

Applications may want to account for this when using multi-pixel fragments; if pixel derivatives are desired, applications should use explicit derivative operations and divide the results by the size of the fragment in each dimension as follows:

$$DPdx(P_n)' = DPdx(P_n) / w$$

$$DPdy(P_n)' = DPdy(P_n) / h$$

where w and h are the size of the fragments in the quad, and $DPdx(P_n)$ and $DPdy(P_n)$ are the pixel derivatives.

The results for `OpDPdx` and `OpDPdy` **may** be calculated as either fine or coarse derivatives, with implementations favoring the most efficient approach. Implementations **must** choose coarse or fine consistently between the two.

Executing `OpFwidthFine`, `OpFwidthCoarse`, or `OpFwidth` is equivalent to executing the corresponding `OpDPdx*` and `OpDPdy*` instructions, taking the absolute value of the results, and summing them.

Executing an `OpImage*Sample*ImplicitLod` instruction is equivalent to executing `OpDPdx(Coordinate)` and `OpDPdy(Coordinate)`, and passing the results as the `Grad` operands `dx` and `dy`.

NOTE

It is expected that using the `ImplicitLod` variants of sampling functions will be substantially more efficient than using the `ExplicitLod` variants with explicitly generated derivatives.

9.19. Helper Invocations

When performing `derivative` or `quad group` operations in a fragment shader, additional invocations **may** be spawned in order to ensure correct results. These additional invocations are known as *helper invocations* and **can** be identified by a non-zero value in the `HelperInvocation` built-in. Stores and atomics performed by helper invocations **must** not have any effect on memory except for the `Function`, `Private` and `Output` storage classes, and values returned by atomic instructions in helper invocations are undefined.

NOTE

While storage to `Output` storage class has an effect even in helper invocations, it does not mean that helper invocations have an effect on the framebuffer. `Output` variables in fragment shaders can be read from as well, and they behave more like `Private` variables for the duration of the shader invocation.

Helper invocations **may** be considered inactive for `group operations` other than `derivative` and `quad group` operations. All invocations in a quad scope instance **may** become permanently inactive at any point once the only remaining invocations in that quad scope instance are helper invocations.

Chapter 10. Pipelines

The following [figure](#) shows a block diagram of the Vulkan pipelines. Some Vulkan commands specify geometric objects to be drawn or computational work to be performed, while others specify state controlling how objects are handled by the various pipeline stages, or control data transfer between memory organized as images and buffers. Commands are effectively sent through a processing pipeline, such as a *graphics pipeline*, or a *compute pipeline*.

The first stage of the [graphics pipeline](#) ([Input Assembler](#)) assembles vertices to form geometric primitives such as points, lines, and triangles, based on a requested primitive topology. In the next stage ([Vertex Shader](#)) vertices **can** be transformed, computing positions and attributes for each vertex. If [tessellation](#) and/or [geometry](#) shaders are supported, they **can** then generate multiple primitives from a single input primitive, possibly changing the primitive topology or generating additional attribute data in the process.

The final resulting primitives are [clipped](#) to a clip volume in preparation for the next stage, [Rasterization](#). The rasterizer produces a series of *fragments* associated with a region of the framebuffer, from a two-dimensional description of a point, line segment, or triangle. These fragments are processed by [fragment operations](#) to determine whether generated values will be written to the framebuffer. [Fragment shading](#) determines the values to be written to the framebuffer attachments. Framebuffer operations then read and write the color and depth/stencil attachments of the framebuffer for a given subpass of a [render pass instance](#). The attachments **can** be used as input attachments in the fragment shader in a later subpass of the same render pass.

The [compute pipeline](#) is a separate pipeline from the graphics pipeline, which operates on one-, two-, or three-dimensional workgroups which **can** read from and write to buffer and image memory.

This ordering is meant only as a tool for describing Vulkan, not as a strict rule of how Vulkan is implemented, and we present it only as a means to organize the various operations of the pipelines. Actual ordering guarantees between pipeline stages are explained in detail in the [synchronization chapter](#).

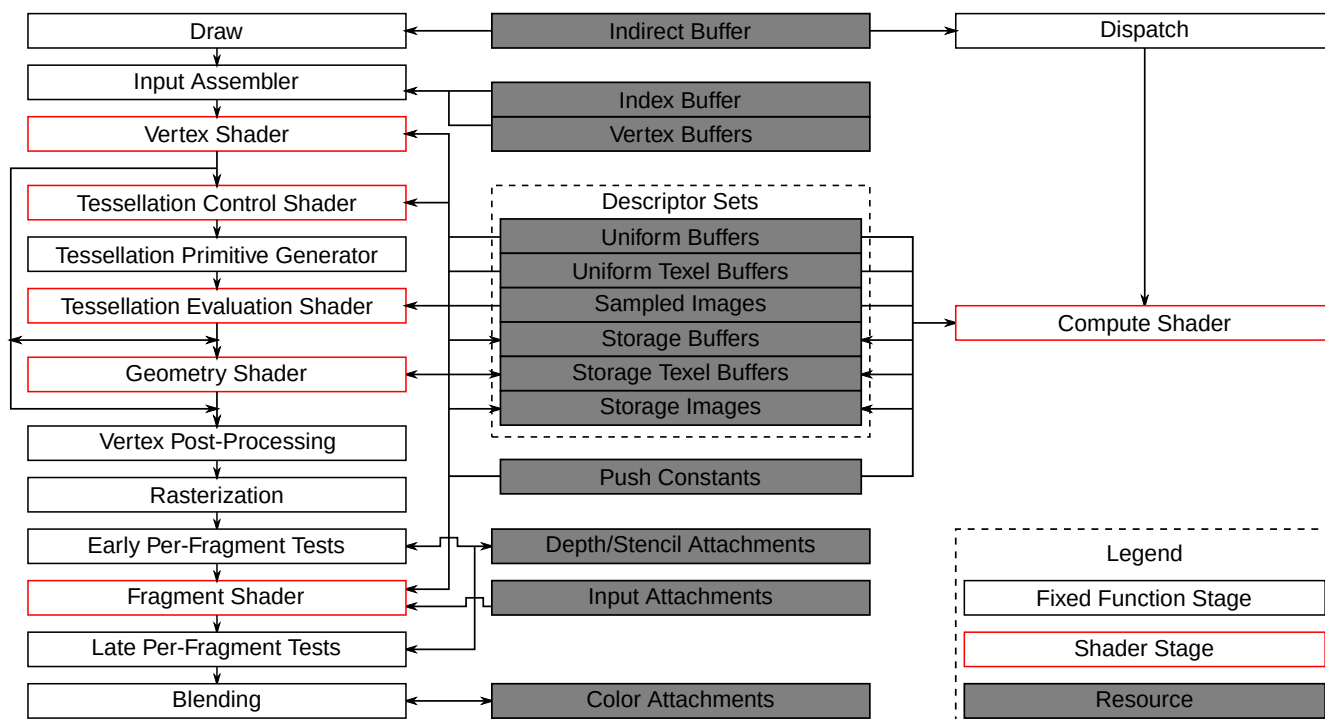


Figure 2. Block diagram of the Vulkan pipeline

Each pipeline is controlled by a monolithic object created from a description of all of the shader stages and any relevant fixed-function stages. [Linking](#) the whole pipeline together allows the optimization of shaders based on their input/outputs and eliminates expensive draw time state validation.

A pipeline object is bound to the current state using [vkCmdBindPipeline](#). Any pipeline object state that is specified as [dynamic](#) is not applied to the current state when the pipeline object is bound, but is instead set by dynamic state setting commands.

No state, including dynamic state, is inherited from one command buffer to another.

Compute, and graphics pipelines are each represented by [VkPipeline](#) handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkPipeline)
```

10.1. Multiple Pipeline Creation

Multiple pipelines **can** be created in a single call by commands such as:

- [vkCreateGraphicsPipelines](#),
- [vkCreateComputePipelines](#).

The creation commands are passed an array [pCreateInfo](#)s of [Vk*PipelineCreateInfo](#) structures specifying parameters of each pipeline to be created, and return a corresponding array of handles in [pPipelines](#). Each element index *i* of [pPipelines](#) is created based on the corresponding element *i* of [pCreateInfo](#)s.

Applications **can** group together similar pipelines to be created in a single call, and implementations are encouraged to look for reuse opportunities when creating a group.

When attempting to create many pipelines in a single command, it is possible that creation **may** fail for a subset of them. In this case, the corresponding elements of `pPipelines` will be `VK_NULL_HANDLE`. If creation fails for a pipeline despite valid arguments (for example, due to out of memory errors), the `VkResult` code returned by the pipeline creation command will indicate why. The implementation will attempt to create all pipelines, and only return `VK_NULL_HANDLE` values for those that actually failed.

If creation fails for multiple pipelines, the returned `VkResult` **must** be the return value of any one of the pipelines which did not succeed. An application **can** reliably clean up from a failed call by iterating over the `pPipelines` array and destroying every element that is not `VK_NULL_HANDLE`.

If the entire command fails and no pipelines are created, all elements of `pPipelines` will be `VK_NULL_HANDLE`.

10.2. Pipeline Creation Flags

When creating a pipeline, the application **can** specify a set of flags that control how the pipeline is created and certain aspects of its behavior and capabilities. These flags are specified by setting the `flags` member of the pipeline creation structure to a bitmask of `VkPipelineCreateFlagBits`.

Bits which **can** be set in

- `VkGraphicsPipelineCreateInfo::flags`
- `VkComputePipelineCreateInfo::flags`

specify how a pipeline is created, and are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPipelineCreateFlagBits {
    VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT = 0x00000001,
    // Provided by VK_VERSION_1_1
    VK_PIPELINE_CREATE_DISPATCH_BASE_BIT = 0x00000010,
    // Provided by VK_VERSION_1_1
    VK_PIPELINE_CREATE_VIEW_INDEX_FROM_DEVICE_INDEX_BIT = 0x00000008,
    // Provided by VK_VERSION_1_1
    // VK_PIPELINE_CREATE_DISPATCH_BASE is a legacy alias
    VK_PIPELINE_CREATE_DISPATCH_BASE = VK_PIPELINE_CREATE_DISPATCH_BASE_BIT,
} VkPipelineCreateFlagBits;
```

- `VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT` specifies that the created pipeline will not be optimized. Using this flag **may** reduce the time taken to create the pipeline.
- `VK_PIPELINE_CREATE_VIEW_INDEX_FROM_DEVICE_INDEX_BIT` specifies that any shader input variables decorated as `ViewIndex` will be assigned values as if they were decorated as `DeviceIndex`.
- `VK_PIPELINE_CREATE_DISPATCH_BASE_BIT` specifies that a compute pipeline **can** be used with

[vkCmdDispatchBase](#) with a non-zero base workgroup.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineCreateFlags;
```

[VkPipelineCreateFlags](#) is a bitmask type for setting a mask of zero or more [VkPipelineCreateFlagBits](#).

10.3. Compute Pipelines

Compute pipelines consist of a single static compute shader stage and the pipeline layout.

The compute pipeline represents a compute shader and is created by calling [vkCreateComputePipelines](#) with an offline compiled pipeline provided in [pipelineCache](#) and the pipeline identified by [VkPipelineOfflineCreateInfo](#) structure in the [pNext](#) chain of [VkComputePipelineCreateInfo](#) structure.

To create compute pipelines, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateComputePipelines(
    VkDevice                device,
    VkPipelineCache         pipelineCache,
    uint32_t                createInfoCount,
    const VkComputePipelineCreateInfo* pCreateInfos,
    const VkAllocationCallbacks* pAllocator,
    VkPipeline*             pPipelines);
```

- [device](#) is the logical device that creates the compute pipelines.
- [pipelineCache](#) is the handle of a valid [VkPipelineCache](#) object. The implementation **must** not access this object outside of the duration of this command.
- [createInfoCount](#) is the length of the [pCreateInfos](#) and [pPipelines](#) arrays.
- [pCreateInfos](#) is a pointer to an array of [VkComputePipelineCreateInfo](#) structures.
- [pAllocator](#) controls host memory allocation as described in the [Memory Allocation](#) chapter.
- [pPipelines](#) is a pointer to an array of [VkPipeline](#) handles in which the resulting compute pipeline objects are returned.

Pipelines are created and returned as described for [Multiple Pipeline Creation](#).

If a pipeline creation fails due to:

- The identified pipeline not being present in [pipelineCache](#)
- The [pNext](#) chain not including a [VkPipelineOfflineCreateInfo](#) structure

the operation will continue as specified in [Multiple Pipeline Creation](#) and the command will return

VK_ERROR_NO_PIPELINE_MATCH.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateComputePipelines` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateComputePipelines-device-09661
`device` **must** support at least one queue family with the `VK_QUEUE_COMPUTE_BIT` capability
- VUID-vkCreateComputePipelines-device-05068
The number of compute pipelines currently allocated from `device` plus `createInfoCount` **must** be less than or equal to the total number of compute pipelines requested via `VkDeviceObjectReservationCreateInfo::computePipelineRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateComputePipelines-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkCreateComputePipelines-pipelineCache-parameter
`pipelineCache` **must** be a valid `VkPipelineCache` handle
- VUID-vkCreateComputePipelines-pCreateInfos-parameter
`pCreateInfos` **must** be a valid pointer to an array of `createInfoCount` valid `VkComputePipelineCreateInfo` structures
- VUID-vkCreateComputePipelines-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateComputePipelines-pPipelines-parameter
`pPipelines` **must** be a valid pointer to an array of `createInfoCount` `VkPipeline` handles
- VUID-vkCreateComputePipelines-createInfoCount-arraylength
`createInfoCount` **must** be greater than `0`
- VUID-vkCreateComputePipelines-pipelineCache-parent
`pipelineCache` **must** have been created, allocated, or retrieved from `device`

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_NO_PIPELINE_MATCH`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

- `VK_ERROR_OUT_OF_POOL_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkComputePipelineCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkComputePipelineCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkPipelineCreateFlags    flags;
    VkPipelineShaderStageCreateInfo stage;
    VkPipelineLayout         layout;
    VkPipeline               basePipelineHandle;
    int32_t                  basePipelineIndex;
} VkComputePipelineCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkPipelineCreateFlagBits` specifying how the pipeline will be generated.
- `stage` is a `VkPipelineShaderStageCreateInfo` structure describing the compute shader.
- `layout` is the description of binding locations used by both the pipeline and descriptor sets used with the pipeline.
- `basePipelineHandle` is a pipeline to derive from. This is not used in Vulkan SC [SCID-8].
- `basePipelineIndex` is an index into the `pCreateInfo` parameter to use as a pipeline to derive from. This is not used in Vulkan SC [SCID-8].

The parameters `basePipelineHandle` and `basePipelineIndex` are described in more detail in [Pipeline Derivatives](#).

Valid Usage

- VUID-VkComputePipelineCreateInfo-None-09497
`flags` **must** be a valid combination of `VkPipelineCreateFlagBits` values
- VUID-VkComputePipelineCreateInfo-basePipelineHandle-05024
`basePipelineHandle` **must** be `VK_NULL_HANDLE`
- VUID-VkComputePipelineCreateInfo-basePipelineIndex-05025
`basePipelineIndex` **must** be zero
- VUID-VkComputePipelineCreateInfo-layout-07987
If a push constant block is declared in a shader, a push constant range in `layout` **must** match the shader stage
- VUID-VkComputePipelineCreateInfo-layout-10069

If a push constant block is declared in a shader, the block must be contained inside the push constant range in **layout** that matches the stage

- VUID-VkComputePipelineCreateInfo-layout-07988
If a **resource variable** is declared in a shader, the corresponding descriptor set in **layout** **must** match the shader stage
- VUID-VkComputePipelineCreateInfo-layout-07990
If a **resource variable** is declared in a shader, the corresponding descriptor set in **layout** **must** match the descriptor type
- VUID-VkComputePipelineCreateInfo-layout-07991
If a **resource variable** is declared in a shader as an array, the corresponding descriptor set in **layout** **must** match the descriptor count
- VUID-VkComputePipelineCreateInfo-stage-00701
The **stage** member of **stage** **must** be **VK_SHADER_STAGE_COMPUTE_BIT**
- VUID-VkComputePipelineCreateInfo-stage-00702
The shader code for the entry point identified by **stage** and the rest of the state identified by this structure **must** adhere to the pipeline linking rules described in the **Shader Interfaces** chapter
- VUID-VkComputePipelineCreateInfo-layout-01687
The number of resources in **layout** accessible to the compute shader stage **must** be less than or equal to **VkPhysicalDeviceLimits::maxPerStageResources**

Valid Usage (Implicit)

- VUID-VkComputePipelineCreateInfo-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_CREATE_INFO**
- VUID-VkComputePipelineCreateInfo-pNext-pNext
pNext **must** be **NULL** or a pointer to a valid instance of **VkPipelineOfflineCreateInfo**
- VUID-VkComputePipelineCreateInfo-sType-unique
The **sType** value of each structure in the **pNext** chain **must** be unique
- VUID-VkComputePipelineCreateInfo-stage-parameter
stage **must** be a valid **VkPipelineShaderStageCreateInfo** structure
- VUID-VkComputePipelineCreateInfo-layout-parameter
layout **must** be a valid **VkPipelineLayout** handle
- VUID-VkComputePipelineCreateInfo-commonparent
Both of **basePipelineHandle**, and **layout** that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same **VkDevice**

The **VkPipelineShaderStageCreateInfo** structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineShaderStageCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkPipelineShaderStageCreateFlags flags;
    VkShaderStageFlagBits    stage;
    VkShaderModule            module;
    const char*              pName;
    const VkSpecializationInfo* pSpecializationInfo;
} VkPipelineShaderStageCreateInfo;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is `NULL` or a pointer to a structure extending this structure.
- **flags** is a bitmask of [VkPipelineShaderStageCreateFlagBits](#) specifying how the pipeline shader stage will be generated.
- **stage** is a [VkShaderStageFlagBits](#) value specifying a single pipeline stage.
- **module** is a [VkShaderModule](#) object containing the shader code for this stage. The implementation **must** not access this object outside of the duration of the command this structure is passed to. This is not used in Vulkan SC [\[SCID-8\]](#).
- **pName** is a pointer to a null-terminated UTF-8 string specifying the entry point name of the shader for this stage.
- **pSpecializationInfo** is a pointer to a [VkSpecializationInfo](#) structure, as described in [Specialization Constants](#), or `NULL`.

In Vulkan SC, the pipeline compilation process occurs [offline](#). Accordingly, **module** **must** be [VK_NULL_HANDLE](#), and the **pName** and **pSpecializationInfo** parameters are not used at runtime and **should** be ignored by the implementation. If provided, the application **must** set the **pName** and **pSpecializationInfo** parameters to the values that were specified for the offline compilation of this pipeline.

Valid Usage

- VUID-VkPipelineShaderStageCreateInfo-stage-00704
If the [geometryShader](#) feature is not enabled, **stage** **must** not be [VK_SHADER_STAGE_GEOMETRY_BIT](#)
- VUID-VkPipelineShaderStageCreateInfo-stage-00705
If the [tessellationShader](#) feature is not enabled, **stage** **must** not be [VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT](#) or [VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT](#)
- VUID-VkPipelineShaderStageCreateInfo-stage-00706
stage **must** not be [VK_SHADER_STAGE_ALL_GRAPHICS](#), or [VK_SHADER_STAGE_ALL](#)
- VUID-VkPipelineShaderStageCreateInfo-module-05026
module **must** be [VK_NULL_HANDLE](#)

- VUID-VkPipelineShaderStageCreateInfo-pName-05027
If `pName` is not `NULL`, it **must** be the name of an `OpEntryPoint` in the SPIR-V shader module used for offline compilation of this pipeline with an execution model that matches `stage`
- VUID-VkPipelineShaderStageCreateInfo-maxClipDistances-00708
If the identified entry point includes any variable in its interface that is declared with the `ClipDistance BuiltIn` decoration, that variable **must** not have an array size greater than `VkPhysicalDeviceLimits::maxClipDistances`
- VUID-VkPipelineShaderStageCreateInfo-maxCullDistances-00709
If the identified entry point includes any variable in its interface that is declared with the `CullDistance BuiltIn` decoration, that variable **must** not have an array size greater than `VkPhysicalDeviceLimits::maxCullDistances`
- VUID-VkPipelineShaderStageCreateInfo-maxCombinedClipAndCullDistances-00710
If the identified entry point includes variables in its interface that are declared with the `ClipDistance BuiltIn` decoration and variables in its interface that are declared with the `CullDistance BuiltIn` decoration, those variables **must** not have array sizes which sum to more than `VkPhysicalDeviceLimits::maxCombinedClipAndCullDistances`
- VUID-VkPipelineShaderStageCreateInfo-maxSampleMaskWords-00711
If the identified entry point includes any variable in its interface that is declared with the `SampleMask BuiltIn` decoration, that variable **must** not have an array size greater than `VkPhysicalDeviceLimits::maxSampleMaskWords`
- VUID-VkPipelineShaderStageCreateInfo-stage-00713
If `stage` is `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` or `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT`, and the identified entry point has an `OpExecutionMode` instruction specifying a patch size with `OutputVertices`, the patch size **must** be greater than `0` and less than or equal to `VkPhysicalDeviceLimits::maxTessellationPatchSize`
- VUID-VkPipelineShaderStageCreateInfo-stage-00714
If `stage` is `VK_SHADER_STAGE_GEOMETRY_BIT`, the identified entry point **must** have an `OpExecutionMode` instruction specifying a maximum output vertex count that is greater than `0` and less than or equal to `VkPhysicalDeviceLimits::maxGeometryOutputVertices`
- VUID-VkPipelineShaderStageCreateInfo-stage-00715
If `stage` is `VK_SHADER_STAGE_GEOMETRY_BIT`, the identified entry point **must** have an `OpExecutionMode` instruction specifying an invocation count that is greater than `0` and less than or equal to `VkPhysicalDeviceLimits::maxGeometryShaderInvocations`
- VUID-VkPipelineShaderStageCreateInfo-stage-02596
If `stage` is either `VK_SHADER_STAGE_VERTEX_BIT`, `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT`, `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT`, or `VK_SHADER_STAGE_GEOMETRY_BIT`, and the identified entry point writes to `Layer` for any primitive, it **must** write the same value to `Layer` for all vertices of a given primitive
- VUID-VkPipelineShaderStageCreateInfo-stage-02597
If `stage` is either `VK_SHADER_STAGE_VERTEX_BIT`, `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT`, `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT`, or `VK_SHADER_STAGE_GEOMETRY_BIT`, and the identified entry point writes to `ViewportIndex` for any primitive, it **must** write the same value to `ViewportIndex` for all vertices of a given primitive

- VUID-VkPipelineShaderStageCreateInfo-stage-06685
If `stage` is `VK_SHADER_STAGE_FRAGMENT_BIT`, and the identified entry point writes to `FragDepth` in any execution path, all execution paths that are not exclusive to helper invocations **must** either discard the fragment, or write or initialize the value of `FragDepth`
- VUID-VkPipelineShaderStageCreateInfo-stage-06686
If `stage` is `VK_SHADER_STAGE_FRAGMENT_BIT`, and the identified entry point writes to `FragStencilRefEXT` in any execution path, all execution paths that are not exclusive to helper invocations **must** either discard the fragment, or write or initialize the value of `FragStencilRefEXT`
- VUID-VkPipelineShaderStageCreateInfo-flags-02784
If `flags` has the `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT` flag set, the `subgroupSizeControl` feature **must** be enabled
- VUID-VkPipelineShaderStageCreateInfo-flags-02785
If `flags` has the `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT` flag set, the `computeFullSubgroups` feature **must** be enabled
- VUID-VkPipelineShaderStageCreateInfo-flags-08988
If `flags` includes `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT`, `stage` **must** be `VK_SHADER_STAGE_COMPUTE_BIT`
- VUID-VkPipelineShaderStageCreateInfo-pNext-02754
If a `VkPipelineShaderStageRequiredSubgroupSizeCreateInfo` structure is included in the `pNext` chain, `flags` **must** not have the `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT` flag set
- VUID-VkPipelineShaderStageCreateInfo-pNext-02755
If a `VkPipelineShaderStageRequiredSubgroupSizeCreateInfo` structure is included in the `pNext` chain, the `subgroupSizeControl` feature **must** be enabled, and `stage` **must** be a valid bit specified in `requiredSubgroupSizeStages`
- VUID-VkPipelineShaderStageCreateInfo-pNext-02756
If a `VkPipelineShaderStageRequiredSubgroupSizeCreateInfo` structure is included in the `pNext` chain and `stage` is `VK_SHADER_STAGE_COMPUTE_BIT`, the local workgroup size of the shader **must** be less than or equal to the product of `VkPipelineShaderStageRequiredSubgroupSizeCreateInfo::requiredSubgroupSize` and `maxComputeWorkgroupSubgroups`
- VUID-VkPipelineShaderStageCreateInfo-pNext-02757
If a `VkPipelineShaderStageRequiredSubgroupSizeCreateInfo` structure is included in the `pNext` chain, and `flags` has the `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT` flag set, the local workgroup size in the X dimension of the pipeline **must** be a multiple of `VkPipelineShaderStageRequiredSubgroupSizeCreateInfo::requiredSubgroupSize`
- VUID-VkPipelineShaderStageCreateInfo-flags-02758
If `flags` has both the `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT` and `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT` flags set, the local workgroup size in the X dimension of the pipeline **must** be a multiple of `maxSubgroupSize`
- VUID-VkPipelineShaderStageCreateInfo-flags-02759
If `flags` has the `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT` flag set and

`flags` does not have the `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT` flag set and no `VkPipelineShaderStageRequiredSubgroupSizeCreateInfo` structure is included in the `pNext` chain, the local workgroup size in the X dimension of the pipeline **must** be a multiple of `subgroupSize`

Valid Usage (Implicit)

- VUID-VkPipelineShaderStageCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO`
- VUID-VkPipelineShaderStageCreateInfo-pNext-pNext
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkDebugUtilsObjectNameInfoEXT` or `VkPipelineShaderStageRequiredSubgroupSizeCreateInfo`
- VUID-VkPipelineShaderStageCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkPipelineShaderStageCreateInfo-flags-parameter
`flags` **must** be a valid combination of `VkPipelineShaderStageCreateFlagBits` values
- VUID-VkPipelineShaderStageCreateInfo-stage-parameter
`stage` **must** be a valid `VkShaderStageFlagBits` value
- VUID-VkPipelineShaderStageCreateInfo-module-parameter
If `module` is not `VK_NULL_HANDLE`, `module` **must** be a valid `VkShaderModule` handle
- VUID-VkPipelineShaderStageCreateInfo-pName-parameter
If `pName` is not `NULL`, `pName` **must** be a null-terminated UTF-8 string
- VUID-VkPipelineShaderStageCreateInfo-pSpecializationInfo-parameter
If `pSpecializationInfo` is not `NULL`, `pSpecializationInfo` **must** be a valid pointer to a valid `VkSpecializationInfo` structure

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineShaderStageCreateFlags;
```

`VkPipelineShaderStageCreateFlags` is a bitmask type for setting a mask of zero or more `VkPipelineShaderStageCreateFlagBits`.

Possible values of the `flags` member of `VkPipelineShaderStageCreateInfo` specifying how a pipeline shader stage is created, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPipelineShaderStageCreateFlagBits {
    VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT = 0x00000001,
    VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT = 0x00000002,
    // Provided by VK_EXT_subgroup_size_control
    VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT =
VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT,
    // Provided by VK_EXT_subgroup_size_control
    VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT =
VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT,
} VkPipelineShaderStageCreateFlagBits;
```

- `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT` specifies that the **SubgroupSize** **may** vary in the shader stage.
- `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT` specifies that the subgroup sizes **must** be launched with all invocations active in the compute stage.

NOTE If `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT` and `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT` are specified and `minSubgroupSize` does not equal `maxSubgroupSize` and no **required subgroup size** is specified, then the only way to guarantee that the 'X' dimension of the local workgroup size is a multiple of **SubgroupSize** is to make it a multiple of **maxSubgroupSize**. Under these conditions, you are guaranteed full subgroups but not any particular subgroup size.

Bits which **can** be set by commands and structures, specifying one or more shader stages, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkShaderStageFlagBits {
    VK_SHADER_STAGE_VERTEX_BIT = 0x00000001,
    VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT = 0x00000002,
    VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT = 0x00000004,
    VK_SHADER_STAGE_GEOMETRY_BIT = 0x00000008,
    VK_SHADER_STAGE_FRAGMENT_BIT = 0x00000010,
    VK_SHADER_STAGE_COMPUTE_BIT = 0x00000020,
    VK_SHADER_STAGE_ALL_GRAPHICS = 0x0000001F,
    VK_SHADER_STAGE_ALL = 0x7FFFFFFF,
} VkShaderStageFlagBits;
```

- `VK_SHADER_STAGE_VERTEX_BIT` specifies the vertex stage.
- `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` specifies the tessellation control stage.
- `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` specifies the tessellation evaluation stage.
- `VK_SHADER_STAGE_GEOMETRY_BIT` specifies the geometry stage.
- `VK_SHADER_STAGE_FRAGMENT_BIT` specifies the fragment stage.
- `VK_SHADER_STAGE_COMPUTE_BIT` specifies the compute stage.

- `VK_SHADER_STAGE_ALL_GRAPHICS` is a combination of bits used as shorthand to specify all graphics stages defined above (excluding the compute stage).
- `VK_SHADER_STAGE_ALL` is a combination of bits used as shorthand to specify all shader stages supported by the device, including all additional stages which are introduced by extensions.

NOTE

`VK_SHADER_STAGE_ALL_GRAPHICS` only includes the original five graphics stages included in Vulkan 1.0, and not any stages added by extensions. Thus, it may not have the desired effect in all cases.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkShaderStageFlags;
```

`VkShaderStageFlags` is a bitmask type for setting a mask of zero or more `VkShaderStageFlagBits`.

The `VkPipelineShaderStageRequiredSubgroupSizeCreateInfo` structure is defined as:

```
typedef struct VkPipelineShaderStageRequiredSubgroupSizeCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           requiredSubgroupSize;
} VkPipelineShaderStageRequiredSubgroupSizeCreateInfo;
```

```
// Provided by VK_EXT_subgroup_size_control
typedef VkPipelineShaderStageRequiredSubgroupSizeCreateInfo
VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `requiredSubgroupSize` is an unsigned integer value specifying the required subgroup size for the newly created pipeline shader stage.

If a `VkPipelineShaderStageRequiredSubgroupSizeCreateInfo` structure is included in the `pNext` chain of `VkPipelineShaderStageCreateInfo`, it specifies that the pipeline shader stage being compiled has a required subgroup size.

Valid Usage

- VUID-VkPipelineShaderStageRequiredSubgroupSizeCreateInfo-requiredSubgroupSize-02760
`requiredSubgroupSize` **must** be a power-of-two integer
- VUID-VkPipelineShaderStageRequiredSubgroupSizeCreateInfo-requiredSubgroupSize-02761
`requiredSubgroupSize` **must** be greater or equal to `minSubgroupSize`

- VUID-VkPipelineShaderStageRequiredSubgroupSizeCreateInfo-requiredSubgroupSize-02762
`requiredSubgroupSize` **must** be less than or equal to `maxSubgroupSize`

Valid Usage (Implicit)

- VUID-VkPipelineShaderStageRequiredSubgroupSizeCreateInfo-sType-sType **must** be `VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_REQUIRED_SUBGROUP_SIZE_CREATE_INFO`

10.4. Graphics Pipelines

Graphics pipelines consist of multiple shader stages, multiple fixed-function pipeline stages, and a pipeline layout.

To create graphics pipelines, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateGraphicsPipelines(
    VkDevice          device,
    VkPipelineCache    pipelineCache,
    uint32_t          createInfoCount,
    const VkGraphicsPipelineCreateInfo* pCreateInfos,
    const VkAllocationCallbacks* pAllocator,
    VkPipeline*        pPipelines);
```

- `device` is the logical device that creates the graphics pipelines.
- `pipelineCache` is the handle of a valid `VkPipelineCache` object. The implementation **must** not access this object outside of the duration of this command.
- `createInfoCount` is the length of the `pCreateInfos` and `pPipelines` arrays.
- `pCreateInfos` is a pointer to an array of `VkGraphicsPipelineCreateInfo` structures.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pPipelines` is a pointer to an array of `VkPipeline` handles in which the resulting graphics pipeline objects are returned.

The `VkGraphicsPipelineCreateInfo` structure includes an array of `VkPipelineShaderStageCreateInfo` structures for each of the desired active shader stages, as well as creation information for all relevant fixed-function stages, and a pipeline layout.

Pipelines are created and returned as described for [Multiple Pipeline Creation](#).

If a pipeline creation fails due to:

- The identified pipeline not being present in `pipelineCache`

- The `pNext` chain not including a `VkPipelineOfflineCreateInfo` structure

the operation will continue as specified in [Multiple Pipeline Creation](#) and the command will return `VK_ERROR_NO_PIPELINE_MATCH`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateGraphicsPipelines` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateGraphicsPipelines-device-09662
`device` **must** support at least one queue family with the `VK_QUEUE_GRAPHICS_BIT` capability
- VUID-vkCreateGraphicsPipelines-device-05068
The number of graphics pipelines currently allocated from `device` plus `createInfoCount` **must** be less than or equal to the total number of graphics pipelines requested via `VkDeviceObjectReservationCreateInfo::graphicsPipelineRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateGraphicsPipelines-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkCreateGraphicsPipelines-pipelineCache-parameter
`pipelineCache` **must** be a valid `VkPipelineCache` handle
- VUID-vkCreateGraphicsPipelines-pCreateInfos-parameter
`pCreateInfos` **must** be a valid pointer to an array of `createInfoCount` valid `VkGraphicsPipelineCreateInfo` structures
- VUID-vkCreateGraphicsPipelines-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateGraphicsPipelines-pPipelines-parameter
`pPipelines` **must** be a valid pointer to an array of `createInfoCount` `VkPipeline` handles
- VUID-vkCreateGraphicsPipelines-createInfoCount-arraylength
`createInfoCount` **must** be greater than `0`
- VUID-vkCreateGraphicsPipelines-pipelineCache-parent
`pipelineCache` **must** have been created, allocated, or retrieved from `device`

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_NO_PIPELINE_MATCH`

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_POOL_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkGraphicsPipelineCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkGraphicsPipelineCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkPipelineCreateFlags     flags;
    uint32_t                  stageCount;
    const VkPipelineShaderStageCreateInfo* pStages;
    const VkPipelineVertexInputStateCreateInfo* pVertexInputState;
    const VkPipelineInputAssemblyStateCreateInfo* pInputAssemblyState;
    const VkPipelineTessellationStateCreateInfo* pTessellationState;
    const VkPipelineViewportStateCreateInfo* pViewportState;
    const VkPipelineRasterizationStateCreateInfo* pRasterizationState;
    const VkPipelineMultisampleStateCreateInfo* pMultisampleState;
    const VkPipelineDepthStencilStateCreateInfo* pDepthStencilState;
    const VkPipelineColorBlendStateCreateInfo* pColorBlendState;
    const VkPipelineDynamicStateCreateInfo* pDynamicState;
    VkPipelineLayout          layout;
    VkRenderPass              renderPass;
    uint32_t                  subpass;
    VkPipeline                basePipelineHandle;
    int32_t                   basePipelineIndex;
} VkGraphicsPipelineCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkPipelineCreateFlagBits` specifying how the pipeline will be generated.
- `stageCount` is the number of entries in the `pStages` array.
- `pStages` is a pointer to an array of `stageCount` `VkPipelineShaderStageCreateInfo` structures describing the set of the shader stages to be included in the graphics pipeline.
- `pVertexInputState` is a pointer to a `VkPipelineVertexInputStateCreateInfo` structure. It **can** be `NULL` if the pipeline is created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state set.
- `pInputAssemblyState` is a pointer to a `VkPipelineInputAssemblyStateCreateInfo` structure which determines input assembly behavior for vertex shading, as described in [Drawing Commands](#).
- `pTessellationState` is a pointer to a `VkPipelineTessellationStateCreateInfo` structure defining tessellation state used by tessellation shaders. It **can** be `NULL` if the pipeline is created with the

`VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT` dynamic state set.

- `pViewportState` is a pointer to a `VkPipelineViewportStateCreateInfo` structure defining viewport state used when rasterization is enabled.
- `pRasterizationState` is a pointer to a `VkPipelineRasterizationStateCreateInfo` structure defining rasterization state.
- `pMultisampleState` is a pointer to a `VkPipelineMultisampleStateCreateInfo` structure defining multisample state used when rasterization is enabled.
- `pDepthStencilState` is a pointer to a `VkPipelineDepthStencilStateCreateInfo` structure defining depth/stencil state used when rasterization is enabled for depth or stencil attachments accessed during rendering.
- `pColorBlendState` is a pointer to a `VkPipelineColorBlendStateCreateInfo` structure defining color blend state used when rasterization is enabled for any color attachments accessed during rendering.
- `pDynamicState` is a pointer to a `VkPipelineDynamicStateCreateInfo` structure defining which properties of the pipeline state object are dynamic and **can** be changed independently of the pipeline state. This **can** be `NULL`, which means no state in the pipeline is considered dynamic.
- `layout` is the description of binding locations used by both the pipeline and descriptor sets used with the pipeline.
- `renderPass` is a handle to a render pass object describing the environment in which the pipeline will be used. The pipeline **must** only be used with a render pass instance compatible with the one provided. See [Render Pass Compatibility](#) for more information. The implementation **must** not access this object outside of the duration of the command this structure is passed to.
- `subpass` is the index of the subpass in the render pass where this pipeline will be used.
- `basePipelineHandle` is a pipeline to derive from. This is not used in Vulkan SC [\[SCID-8\]](#).
- `basePipelineIndex` is an index into the `pCreateInfos` parameter to use as a pipeline to derive from. This is not used in Vulkan SC [\[SCID-8\]](#).

The parameters `basePipelineHandle` and `basePipelineIndex` are described in more detail in [Pipeline Derivatives](#).

The state required for a graphics pipeline is divided into [vertex input state](#), [pre-rasterization shader state](#), [fragment shader state](#), and [fragment output state](#).

Vertex Input State

Vertex input state is defined by:

- [VkPipelineVertexInputStateCreateInfo](#)
- [VkPipelineInputAssemblyStateCreateInfo](#)

This state **must** be specified to create a [complete graphics pipeline](#).

Pre-Rasterization Shader State

Pre-rasterization shader state is defined by:

- [VkPipelineShaderStageCreateInfo](#) entries for:
 - Vertex shaders
 - Tessellation control shaders
 - Tessellation evaluation shaders
 - Geometry shaders
- Within the [VkPipelineLayout](#), the full pipeline layout **must** be specified.
- [VkPipelineViewportStateCreateInfo](#)
- [VkPipelineRasterizationStateCreateInfo](#)
- [VkPipelineTessellationStateCreateInfo](#)
- [VkRenderPass](#) and **subpass** parameter
- [VkPipelineDiscardRectangleStateCreateInfoEXT](#)
- [VkPipelineFragmentShadingRateStateCreateInfoKHR](#)

This state **must** be specified to create a [complete graphics pipeline](#).

Fragment Shader State

Fragment shader state is defined by:

- A [VkPipelineShaderStageCreateInfo](#) entry for the fragment shader
- Within the [VkPipelineLayout](#), the full pipeline layout **must** be specified.
- [VkPipelineMultisampleStateCreateInfo](#)
- [VkPipelineDepthStencilStateCreateInfo](#)
- [VkRenderPass](#) and **subpass** parameter
- [VkPipelineFragmentShadingRateStateCreateInfoKHR](#)

If **rasterizerDiscardEnable** is **VK_FALSE** or **VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE** is used, this state **must** be specified to create a [complete graphics pipeline](#).

Fragment Output State

Fragment output state is defined by:

- [VkPipelineColorBlendStateCreateInfo](#)
- [VkRenderPass](#) and **subpass** parameter
- [VkPipelineMultisampleStateCreateInfo](#)

If **rasterizerDiscardEnable** is **VK_FALSE** or **VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE** is used, this state **must** be specified to create a [complete graphics pipeline](#).

Dynamic State

Dynamic state values set via **pDynamicState** **must** be ignored if the state they correspond to is not otherwise statically set by one of the state subsets used to create the pipeline. For example, if a pipeline only included [pre-rasterization shader state](#), then any dynamic state value corresponding

to depth or stencil testing has no effect.

Complete Graphics Pipelines

A complete graphics pipeline always includes [pre-rasterization shader state](#), with other subsets included depending on that state as specified in the above sections.

In Vulkan SC, the pipeline compilation process occurs [offline](#) and the [pStages](#) are not needed at runtime and **may** be omitted. If omitted, [stageCount](#) **must** be 0 and [pStages](#) **must** be `NULL`. If provided, the values **must** match the values specified to the offline compiler.

Valid Usage

- VUID-VkGraphicsPipelineCreateInfo-None-09497
[flags](#) **must** be a valid combination of [VkPipelineCreateFlagBits](#) values
- VUID-VkGraphicsPipelineCreateInfo-basePipelineHandle-05024
[basePipelineHandle](#) **must** be `VK_NULL_HANDLE`
- VUID-VkGraphicsPipelineCreateInfo-basePipelineIndex-05025
[basePipelineIndex](#) **must** be zero
- VUID-VkGraphicsPipelineCreateInfo-layout-07987
If a push constant block is declared in a shader, a push constant range in [layout](#) **must** match the shader stage
- VUID-VkGraphicsPipelineCreateInfo-layout-10069
If a push constant block is declared in a shader, the block must be contained inside the push constant range in [layout](#) that matches the stage
- VUID-VkGraphicsPipelineCreateInfo-layout-07988
If a [resource variable](#) is declared in a shader, the corresponding descriptor set in [layout](#) **must** match the shader stage
- VUID-VkGraphicsPipelineCreateInfo-layout-07990
If a [resource variable](#) is declared in a shader, the corresponding descriptor set in [layout](#) **must** match the descriptor type
- VUID-VkGraphicsPipelineCreateInfo-layout-07991
If a [resource variable](#) is declared in a shader as an array, the corresponding descriptor set in [layout](#) **must** match the descriptor count
- VUID-VkGraphicsPipelineCreateInfo-stage-02096
If the pipeline requires [pre-rasterization shader state](#) the [stage](#) member of one element of [pStages](#) **must** be `VK_SHADER_STAGE_VERTEX_BIT`
- VUID-VkGraphicsPipelineCreateInfo-pStages-00729
If the pipeline requires [pre-rasterization shader state](#) and [pStages](#) includes a tessellation control shader stage, it **must** include a tessellation evaluation shader stage
- VUID-VkGraphicsPipelineCreateInfo-pStages-00730
If the pipeline requires [pre-rasterization shader state](#) and [pStages](#) includes a tessellation evaluation shader stage, it **must** include a tessellation control shader stage
- VUID-VkGraphicsPipelineCreateInfo-pStages-09022

If the pipeline requires [pre-rasterization shader state](#) and [pStages](#) includes a tessellation control shader stage, [pTessellationState](#) **must** be a valid pointer to a valid [VkPipelineTessellationStateCreateInfo](#) structure

- VUID-VkGraphicsPipelineCreateInfo-pStages-00732

If the pipeline requires [pre-rasterization shader state](#) and [pStages](#) includes tessellation shader stages, the shader code of at least one stage **must** contain an [OpExecutionMode](#) instruction specifying the type of subdivision in the pipeline

- VUID-VkGraphicsPipelineCreateInfo-pStages-00733

If the pipeline requires [pre-rasterization shader state](#) and [pStages](#) includes tessellation shader stages, and the shader code of both stages contain an [OpExecutionMode](#) instruction specifying the type of subdivision in the pipeline, they **must** both specify the same subdivision mode

- VUID-VkGraphicsPipelineCreateInfo-pStages-00734

If the pipeline requires [pre-rasterization shader state](#) and [pStages](#) includes tessellation shader stages, the shader code of at least one stage **must** contain an [OpExecutionMode](#) instruction specifying the output patch size in the pipeline

- VUID-VkGraphicsPipelineCreateInfo-pStages-00735

If the pipeline requires [pre-rasterization shader state](#) and [pStages](#) includes tessellation shader stages, and the shader code of both contain an [OpExecutionMode](#) instruction specifying the out patch size in the pipeline, they **must** both specify the same patch size

- VUID-VkGraphicsPipelineCreateInfo-pStages-08888

If the pipeline is being created with [pre-rasterization shader state](#) and [vertex input state](#) and [pStages](#) includes tessellation shader stages, the [topology](#) member of [pInputAssembly](#) **must** be [VK_PRIMITIVE_TOPOLOGY_PATCH_LIST](#)

- VUID-VkGraphicsPipelineCreateInfo-topology-08889

If the pipeline is being created with [pre-rasterization shader state](#) and [vertex input state](#) and the [topology](#) member of [pInputAssembly](#) is [VK_PRIMITIVE_TOPOLOGY_PATCH_LIST](#), then [pStages](#) **must** include tessellation shader stages

- VUID-VkGraphicsPipelineCreateInfo-TessellationEvaluation-07723

If the pipeline is being created with a [TessellationEvaluation Execution Model](#), no [Geometry Execution Model](#), uses the [PointMode Execution Mode](#), and the [shaderTessellationAndGeometryPointSize](#) feature is enabled, a [PointSize](#) decorated variable **must** be written to

- VUID-VkGraphicsPipelineCreateInfo-topology-08773

If the pipeline is being created with a [Vertex Execution Model](#) and no [TessellationEvaluation](#) or [Geometry Execution Model](#), and the [topology](#) member of [pInputAssembly](#) is [VK_PRIMITIVE_TOPOLOGY_POINT_LIST](#), a [PointSize](#) decorated variable **must** be written to

- VUID-VkGraphicsPipelineCreateInfo-TessellationEvaluation-07724

If the pipeline is being created with a [TessellationEvaluation Execution Model](#), no [Geometry Execution Model](#), uses the [PointMode Execution Mode](#), and the [shaderTessellationAndGeometryPointSize](#) feature is not enabled, a [PointSize](#) decorated variable **must** not be written to

- VUID-VkGraphicsPipelineCreateInfo-shaderTessellationAndGeometryPointSize-08776

If the pipeline is being created with a **Geometry Execution Model**, uses the **OutputPoints Execution Mode**, and the **shaderTessellationAndGeometryPointSize** feature is enabled, a **PointSize** decorated variable **must** be written to for every vertex emitted

- VUID-VkGraphicsPipelineCreateInfo-Geometry-07726

If the pipeline is being created with a **Geometry Execution Model**, uses the **OutputPoints Execution Mode**, and the **shaderTessellationAndGeometryPointSize** feature is not enabled, a **PointSize** decorated variable **must** not be written to

- VUID-VkGraphicsPipelineCreateInfo-pStages-00738

If the pipeline requires **pre-rasterization shader state** and **pStages** includes a geometry shader stage, and does not include any tessellation shader stages, its shader code **must** contain an **OpExecutionMode** instruction specifying an input primitive type that is **compatible** with the primitive topology specified in **pInputAssembly**

- VUID-VkGraphicsPipelineCreateInfo-pStages-00739

If the pipeline requires **pre-rasterization shader state** and **pStages** includes a geometry shader stage, and also includes tessellation shader stages, its shader code **must** contain an **OpExecutionMode** instruction specifying an input primitive type that is **compatible** with the primitive topology that is output by the tessellation stages

- VUID-VkGraphicsPipelineCreateInfo-pStages-00740

If the pipeline requires **pre-rasterization shader state** and **fragment shader state**, it includes both a fragment shader and a geometry shader, and the fragment shader code reads from an input variable that is decorated with **PrimitiveId**, then the geometry shader code **must** write to a matching output variable, decorated with **PrimitiveId**, in all execution paths

- VUID-VkGraphicsPipelineCreateInfo-renderPass-06038

If **renderPass** is not **VK_NULL_HANDLE** and the pipeline is being created with **fragment shader state** the fragment shader **must** not read from any input attachment that is defined as **VK_ATTACHMENT_UNUSED** in **subpass**

- VUID-VkGraphicsPipelineCreateInfo-pStages-00742

If the pipeline requires **pre-rasterization shader state** and multiple pre-rasterization shader stages are included in **pStages**, the shader code for the entry points identified by those **pStages** and the rest of the state identified by this structure **must** adhere to the pipeline linking rules described in the **Shader Interfaces** chapter

- VUID-VkGraphicsPipelineCreateInfo-None-04889

If the pipeline requires **pre-rasterization shader state** and **fragment shader state**, the fragment shader and last **pre-rasterization shader stage** and any relevant state **must** adhere to the pipeline linking rules described in the **Shader Interfaces** chapter

- VUID-VkGraphicsPipelineCreateInfo-renderPass-06041

If **renderPass** is not **VK_NULL_HANDLE**, and the pipeline is being created with **fragment output interface state**, then for each color attachment in the subpass, if the **potential format features** of the format of the corresponding attachment description do not contain **VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT**, then the **blendEnable** member of the corresponding element of the **pAttachments** member of **pColorBlendState** **must** be **VK_FALSE**

- VUID-VkGraphicsPipelineCreateInfo-renderPass-07609

If **renderPass** is not **VK_NULL_HANDLE**, the pipeline is being created with **fragment output**

interface state, the `pColorBlendState` pointer is not `NULL`, the `attachmentCount` member of `pColorBlendState` is not ignored, and the subpass uses color attachments, the `attachmentCount` member of `pColorBlendState` **must** be equal to the `colorAttachmentCount` used to create subpass

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04130
If the pipeline requires `pre-rasterization shader state`, and `pViewportState->pViewports` is not dynamic, then `pViewportState->pViewports` **must** be a valid pointer to an array of `pViewportState->viewportCount` valid `VkViewport` structures
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04131
If the pipeline requires `pre-rasterization shader state`, and `pViewportState->pScissors` is not dynamic, then `pViewportState->pScissors` **must** be a valid pointer to an array of `pViewportState->scissorCount` `VkRect2D` structures
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-00749
If the pipeline requires `pre-rasterization shader state`, and the `wideLines` feature is not enabled, and no element of the `pDynamicStates` member of `pDynamicState` is `VK_DYNAMIC_STATE_LINE_WIDTH`, the `lineWidth` member of `pRasterizationState` **must** be `1.0`
- VUID-VkGraphicsPipelineCreateInfo-rasterizerDiscardEnable-09024
If the pipeline requires `pre-rasterization shader state`, and the `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` dynamic state is enabled or the `rasterizerDiscardEnable` member of `pRasterizationState` is `VK_FALSE`, `pViewportState` **must** be a valid pointer to a valid `VkPipelineViewportStateCreateInfo` structure
- VUID-VkGraphicsPipelineCreateInfo-pMultisampleState-09026
If the pipeline requires `fragment output interface state`, `pMultisampleState` **must** be a valid pointer to a valid `VkPipelineMultisampleStateCreateInfo` structure
- VUID-VkGraphicsPipelineCreateInfo-pMultisampleState-09027
If `pMultisampleState` is not `NULL` it **must** be a valid pointer to a valid `VkPipelineMultisampleStateCreateInfo` structure
- VUID-VkGraphicsPipelineCreateInfo-alphaToCoverageEnable-08891
If the pipeline is being created with `fragment shader state`, the `VkPipelineMultisampleStateCreateInfo::alphaToCoverageEnable` is not ignored and is `VK_TRUE`, then the `Fragment Output Interface` **must** contain a variable for the alpha Component word in Location 0 at Index 0
- VUID-VkGraphicsPipelineCreateInfo-renderPass-09028
If `renderPass` is not `VK_NULL_HANDLE`, the pipeline is being created with `fragment shader state`, and subpass uses a depth/stencil attachment, `pDepthStencilState` **must** be a valid pointer to a valid `VkPipelineDepthStencilStateCreateInfo` structure
- VUID-VkGraphicsPipelineCreateInfo-pDepthStencilState-09029
If `pDepthStencilState` is not `NULL` it **must** be a valid pointer to a valid `VkPipelineDepthStencilStateCreateInfo` structure
- VUID-VkGraphicsPipelineCreateInfo-renderPass-09030
If `renderPass` is not `VK_NULL_HANDLE`, the pipeline is being created with `fragment output interface state`, and subpass uses color attachments, `pColorBlendState` **must** be a valid pointer to a valid `VkPipelineColorBlendStateCreateInfo` structure

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-00754
If the pipeline requires [pre-rasterization shader state](#), the [depthBiasClamp](#) feature is not enabled, no element of the [pDynamicStates](#) member of [pDynamicState](#) is [VK_DYNAMIC_STATE_DEPTH_BIAS](#), and the [depthBiasEnable](#) member of [pRasterizationState](#) is [VK_TRUE](#), the [depthBiasClamp](#) member of [pRasterizationState](#) **must** be [0.0](#)
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-02510
If the pipeline requires [fragment shader state](#), the [VK_EXT_depth_range_unrestricted](#) extension is not enabled and no element of the [pDynamicStates](#) member of [pDynamicState](#) is [VK_DYNAMIC_STATE_DEPTH_BOUNDS](#), and the [depthBoundsTestEnable](#) member of [pDepthStencilState](#) is [VK_TRUE](#), the [minDepthBounds](#) and [maxDepthBounds](#) members of [pDepthStencilState](#) **must** be between [0.0](#) and [1.0](#), inclusive
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-10913
If the pipeline requires [fragment shader state](#), no element of the [pDynamicStates](#) member of [pDynamicState](#) is [VK_DYNAMIC_STATE_DEPTH_BOUNDS](#), and [pDynamicStates](#) includes [VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE](#) or the [depthBoundsTestEnable](#) member of [pDepthStencilState](#) is [VK_TRUE](#), [minDepthBounds](#) **must** be less than or equal to [maxDepthBounds](#)
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07610
If the pipeline requires [fragment shader state](#) or [fragment output interface state](#), and [rasterizationSamples](#) and [sampleLocationsInfo](#) are not dynamic, and [VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable](#) included in the [pNext](#) chain of [pMultisampleState](#) is [VK_TRUE](#), [sampleLocationsInfo.sampleLocationGridSize.width](#) **must** evenly divide [VkMultisamplePropertiesEXT::maxSampleLocationGridSize.width](#) as returned by [vkGetPhysicalDeviceMultisamplePropertiesEXT](#) with a [samples](#) parameter equaling [rasterizationSamples](#)
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07611
If the pipeline requires [fragment shader state](#) or [fragment output interface state](#), and [rasterizationSamples](#) and [sampleLocationsInfo](#) are not dynamic, and [VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable](#) the included in the [pNext](#) chain of [pMultisampleState](#) is [VK_TRUE](#) or [VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT](#) is used, [sampleLocationsInfo.sampleLocationGridSize.height](#) **must** evenly divide [VkMultisamplePropertiesEXT::maxSampleLocationGridSize.height](#) as returned by [vkGetPhysicalDeviceMultisamplePropertiesEXT](#) with a [samples](#) parameter equaling [rasterizationSamples](#)
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07612
If the pipeline requires [fragment shader state](#) or [fragment output interface state](#), and [rasterizationSamples](#) and [sampleLocationsInfo](#) are not dynamic, and [VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable](#) included in the [pNext](#) chain of [pMultisampleState](#) is [VK_TRUE](#) or [VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_ENABLE_EXT](#) is used, [sampleLocationsInfo.sampleLocationsPerPixel](#) **must** equal [rasterizationSamples](#)
- VUID-VkGraphicsPipelineCreateInfo-sampleLocationsEnable-01524
If the pipeline requires [fragment shader state](#), and the [sampleLocationsEnable](#) member of a [VkPipelineSampleLocationsStateCreateInfoEXT](#) structure included in the [pNext](#) chain of

`pMultisampleState` is `VK_TRUE`, the fragment shader code **must** not statically use the extended instruction `InterpolateAtSample`

- VUID-VkGraphicsPipelineCreateInfo-subpass-00758
If the pipeline requires `fragment output interface state`, `rasterizationSamples` is not dynamic, and `subpass` does not use any color and/or depth/stencil attachments, then the `rasterizationSamples` member of `pMultisampleState` **must** follow the rules for a `zero-attachment subpass`
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06046
If `renderPass` is not `VK_NULL_HANDLE`, `subpass` **must** be a valid subpass within `renderPass`
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06047
If `renderPass` is not `VK_NULL_HANDLE`, the pipeline is being created with `pre-rasterization shader state`, `subpass` `viewMask` is not `0`, and `multiviewTessellationShader` is not enabled, then `pStages` **must** not include tessellation shaders
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06048
If `renderPass` is not `VK_NULL_HANDLE`, the pipeline is being created with `pre-rasterization shader state`, `subpass` `viewMask` is not `0`, and `multiviewGeometryShader` is not enabled, then `pStages` **must** not include a geometry shader
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06050
If `renderPass` is not `VK_NULL_HANDLE` and the pipeline is being created with `pre-rasterization shader state`, and `subpass` `viewMask` is not `0`, then all of the shaders in the pipeline **must** not include variables decorated with the `Layer` built-in decoration in their interfaces
- VUID-VkGraphicsPipelineCreateInfo-flags-00764
`flags` **must** not contain the `VK_PIPELINE_CREATE_DISPATCH_BASE_BIT` flag
- VUID-VkGraphicsPipelineCreateInfo-pStages-01565
If the pipeline requires `fragment shader state` and an input attachment was referenced by an `aspectMask` at `renderPass` creation time, the fragment shader **must** only read from the aspects that were specified for that input attachment
- VUID-VkGraphicsPipelineCreateInfo-layout-01688
The number of resources in `layout` accessible to each shader stage that is used by the pipeline **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageResources`
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04058
If the pipeline requires `pre-rasterization shader state`, and no element of the `pDynamicStates` member of `pDynamicState` is `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT`, and if `pNext` chain includes a `VkPipelineDiscardRectangleStateCreateInfoEXT` structure, and if its `discardRectangleCount` member is not `0`, then its `pDiscardRectangles` member **must** be a valid pointer to an array of `discardRectangleCount` `VkRect2D` structures
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07855
If `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT` is included in the `pDynamicStates` array then the implementation **must** support at least `specVersion 2` of the `VK_EXT_discard_rectangles` extension
- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-07856
If `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT` is included in the `pDynamicStates` array

then the implementation **must** support at least `specVersion 2` of the `VK_EXT_discard_rectangles` extension

- VUID-VkGraphicsPipelineCreateInfo-pStages-02097

If the pipeline requires `vertex input state`, and `pVertexInputState` is not dynamic, then `pVertexInputState` **must** be a valid pointer to a valid `VkPipelineVertexInputStateCreateInfo` structure

- VUID-VkGraphicsPipelineCreateInfo-Input-07904

If the pipeline is being created with `vertex input state` and `pVertexInputState` is not dynamic, then all variables with the `Input` storage class decorated with `Location` in the `Vertex Execution Model OpEntryPoint` **must** contain a location in `VkVertexInputAttributeDescription::location`

- VUID-VkGraphicsPipelineCreateInfo-Input-08733

If the pipeline requires `vertex input state` and `pVertexInputState` is not dynamic, then the numeric type associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be the same as `VkVertexInputAttributeDescription::format`

- VUID-VkGraphicsPipelineCreateInfo-pVertexInputState-08929

If the pipeline is being created with `vertex input state` and `pVertexInputState` is not dynamic, and `VkVertexInputAttributeDescription::format` has a 64-bit component, then the scalar width associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be 64-bit

- VUID-VkGraphicsPipelineCreateInfo-pVertexInputState-08930

If the pipeline is being created with `vertex input state` and `pVertexInputState` is not dynamic, and the scalar width associated with a `Location` decorated `Input` variable in the `Vertex Execution Model OpEntryPoint` is 64-bit, then the corresponding `VkVertexInputAttributeDescription::format` **must** have a 64-bit component

- VUID-VkGraphicsPipelineCreateInfo-pVertexInputState-09198

If the pipeline is being created with `vertex input state` and `pVertexInputState` is not dynamic, and `VkVertexInputAttributeDescription::format` has a 64-bit component, then all `Input` variables at the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** not use components that are not present in the format

- VUID-VkGraphicsPipelineCreateInfo-dynamicPrimitiveTopologyUnrestricted-09031

If the pipeline requires `vertex input state`, `pInputAssemblyState` **must** be a valid pointer to a valid `VkPipelineInputAssemblyStateCreateInfo` structure

- VUID-VkGraphicsPipelineCreateInfo-pInputAssemblyState-09032

If `pInputAssemblyState` is not `NULL` it **must** be a valid pointer to a valid `VkPipelineInputAssemblyStateCreateInfo` structure

- VUID-VkGraphicsPipelineCreateInfo-lineRasterizationMode-02766

If the pipeline requires `pre-rasterization shader state` and at least one of `fragment output interface state` or `fragment shader state`, and `pMultisampleState` is not `NULL`, the `lineRasterizationMode` member of a `VkPipelineRasterizationLineStateCreateInfo` structure included in the `pNext` chain of `pRasterizationState` is `VK_LINE_RASTERIZATION_MODE_BRESENHAM` or `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH`, then the `alphaToCoverageEnable`, `alphaToOneEnable`, and `sampleShadingEnable` members of

`pMultisampleState` **must** all be `VK_FALSE`

- VUID-VkGraphicsPipelineCreateInfo-stippledLineEnable-02767

If the pipeline requires `pre-rasterization shader state`, the `stippledLineEnable` member of `VkPipelineRasterizationLineStateCreateInfo` is `VK_TRUE`, and no element of the `pDynamicStates` member of `pDynamicState` is `VK_DYNAMIC_STATE_LINE_STIPPLE`, then the `lineStippleFactor` member of `VkPipelineRasterizationLineStateCreateInfo` **must** be in the range [1,256]

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-03378

If the `extendedDynamicState` feature is not enabled, there **must** be no element of the `pDynamicStates` member of `pDynamicState` set to `VK_DYNAMIC_STATE_CULL_MODE`, `VK_DYNAMIC_STATE_FRONT_FACE`, `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY`, `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT`, `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT`, `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE`, `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE`, `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE`, `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP`, `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE`, `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE`, or `VK_DYNAMIC_STATE_STENCIL_OP`

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-03379

If the pipeline requires `pre-rasterization shader state`, and `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` is included in the `pDynamicStates` array then `viewportCount` **must** be zero

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-03380

If the pipeline requires `pre-rasterization shader state`, and `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` is included in the `pDynamicStates` array then `scissorCount` **must** be zero

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04132

If the pipeline requires `pre-rasterization shader state`, and `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` is included in the `pDynamicStates` array then `VK_DYNAMIC_STATE_VIEWPORT` **must** not be present

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04133

If the pipeline requires `pre-rasterization shader state`, and `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` is included in the `pDynamicStates` array then `VK_DYNAMIC_STATE_SCISSOR` **must** not be present

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04868

If the `extendedDynamicState2` feature is not enabled, there **must** be no element of the `pDynamicStates` member of `pDynamicState` set to `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE`, `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE`, or `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE`

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04869

If the `extendedDynamicState2LogicOp` feature is not enabled, there **must** be no element of the `pDynamicStates` member of `pDynamicState` set to `VK_DYNAMIC_STATE_LOGIC_OP_EXT`

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04870

If the `extendedDynamicState2PatchControlPoints` feature is not enabled, there **must** be no element of the `pDynamicStates` member of `pDynamicState` set to `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT`

- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04494
If the pipeline requires [pre-rasterization shader state](#) or [fragment shader state](#) and [VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR](#) is not included in `pDynamicState->pDynamicStates`, [VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.width](#) **must** be greater than or equal to 1
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04495
If the pipeline requires [pre-rasterization shader state](#) or [fragment shader state](#) and [VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR](#) is not included in `pDynamicState->pDynamicStates`, [VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.height](#) **must** be greater than or equal to 1
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04496
If the pipeline requires [pre-rasterization shader state](#) or [fragment shader state](#) and [VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR](#) is not included in `pDynamicState->pDynamicStates`, [VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.width](#) **must** be a power-of-two value
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04497
If the pipeline requires [pre-rasterization shader state](#) or [fragment shader state](#) and [VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR](#) is not included in `pDynamicState->pDynamicStates`, [VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.height](#) **must** be a power-of-two value
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04498
If the pipeline requires [pre-rasterization shader state](#) or [fragment shader state](#) and [VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR](#) is not included in `pDynamicState->pDynamicStates`, [VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.width](#) **must** be less than or equal to 4
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04499
If the pipeline requires [pre-rasterization shader state](#) or [fragment shader state](#) and [VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR](#) is not included in `pDynamicState->pDynamicStates`, [VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.height](#) **must** be less than or equal to 4
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04500
If the pipeline requires [pre-rasterization shader state](#) or [fragment shader state](#) and [VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR](#) is not included in `pDynamicState->pDynamicStates`, and the [pipelineFragmentShadingRate](#) feature is not enabled, [VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.width](#) and [VkPipelineFragmentShadingRateStateCreateInfoKHR::fragmentSize.height](#) **must** both be equal to 1
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-06567
If the pipeline requires [pre-rasterization shader state](#) or [fragment shader state](#) and [VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR](#) is not included in `pDynamicState->pDynamicStates`, [VkPipelineFragmentShadingRateStateCreateInfoKHR::combinerOps\[0\]](#) **must** be a valid [VkFragmentShadingRateCombinerOpKHR](#) value
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-06568
If the pipeline requires [pre-rasterization shader state](#) or [fragment shader state](#) and [VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR](#) is not included in `pDynamicState->pDynamicStates`, [VkPipelineFragmentShadingRateStateCreateInfoKHR::combinerOps\[0\]](#) **must** be a valid [VkFragmentShadingRateCombinerOpKHR](#) value

`>pDynamicStates`, `VkPipelineFragmentShadingRateStateCreateInfoKHR::combinerOps[1]` **must** be a valid `VkFragmentShadingRateCombinerOpKHR` value

- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04501

If the pipeline requires `pre-rasterization shader state` or `fragment shader state` and `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` is not included in `pDynamicState->pDynamicStates`, and the `primitiveFragmentShadingRate` feature is not enabled, `VkPipelineFragmentShadingRateStateCreateInfoKHR::combinerOps[0]` **must** be `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR`

- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-04502

If the pipeline requires `pre-rasterization shader state` or `fragment shader state` and `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` is not included in `pDynamicState->pDynamicStates`, and the `attachmentFragmentShadingRate` feature is not enabled, `VkPipelineFragmentShadingRateStateCreateInfoKHR::combinerOps[1]` **must** be `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR`

- VUID-VkGraphicsPipelineCreateInfo-primitiveFragmentShadingRateWithMultipleViewports-04503

If the pipeline requires `pre-rasterization shader state` and the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` is not included in `pDynamicState->pDynamicStates`, and `VkPipelineViewportStateCreateInfo::viewportCount` is greater than 1, entry points specified in `pStages` **must** not write to the `PrimitiveShadingRateKHR` built-in

- VUID-VkGraphicsPipelineCreateInfo-primitiveFragmentShadingRateWithMultipleViewports-04504

If the pipeline requires `pre-rasterization shader state` and the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, and entry points specified in `pStages` write to the `ViewportIndex` built-in, they **must** not also write to the `PrimitiveShadingRateKHR` built-in

- VUID-VkGraphicsPipelineCreateInfo-fragmentShadingRateNonTrivialCombinerOps-04506

If the pipeline requires `pre-rasterization shader state` or `fragment shader state`, the `fragmentShadingRateNonTrivialCombinerOps` limit is not supported, and `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` is not included in `pDynamicState->pDynamicStates`, elements of `VkPipelineFragmentShadingRateStateCreateInfoKHR::combinerOps` **must** be `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR` or `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_REPLACE_KHR`

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04807

If the pipeline requires `pre-rasterization shader state` and the `vertexInputDynamicState` feature is not enabled, there **must** be no element of the `pDynamicStates` member of `pDynamicState` set to `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT`

- VUID-VkGraphicsPipelineCreateInfo-pDynamicStates-04800

If the `colorWriteEnable` feature is not enabled, there **must** be no element of the `pDynamicStates` member of `pDynamicState` set to `VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT`

- VUID-VkGraphicsPipelineCreateInfo-pStages-06600

If the pipeline requires `pre-rasterization shader state` or `fragment shader state`, `pStages` **must** be a valid pointer to an array of `stageCount` valid `VkPipelineShaderStageCreateInfo` structures

- VUID-VkGraphicsPipelineCreateInfo-stageCount-09587
If the pipeline does not require [pre-rasterization shader state](#) or [fragment shader state](#), **stageCount** **must** be zero
- VUID-VkGraphicsPipelineCreateInfo-pRasterizationState-06601
If the pipeline requires [pre-rasterization shader state](#), **pRasterizationState** **must** be a valid pointer to a valid [VkPipelineRasterizationStateCreateInfo](#) structure
- VUID-VkGraphicsPipelineCreateInfo-layout-06602
If the pipeline requires [fragment shader state](#) or [pre-rasterization shader state](#), **layout** **must** be a valid [VkPipelineLayout](#) handle
- VUID-VkGraphicsPipelineCreateInfo-renderPass-06603
If the pipeline requires [pre-rasterization shader state](#), [fragment shader state](#), or [fragment output state](#), **renderPass** **must** be a valid [VkRenderPass](#) handle
- VUID-VkGraphicsPipelineCreateInfo-stageCount-09530
If the pipeline requires [pre-rasterization shader state](#), **stageCount** **must** be greater than 0
- VUID-VkGraphicsPipelineCreateInfo-conservativePointAndLineRasterization-08892
If **conservativePointAndLineRasterization** is not supported and the [effective rasterization input topology](#) is in line or point topology class, then [VkPipelineRasterizationConservativeStateCreateInfoEXT::conservativeRasterizationMode](#) **must** be **VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT**
- VUID-VkGraphicsPipelineCreateInfo-pStages-06894
If the pipeline requires [pre-rasterization shader state](#) but not [fragment shader state](#), elements of **pStages** **must** not have **stage** set to **VK_SHADER_STAGE_FRAGMENT_BIT**
- VUID-VkGraphicsPipelineCreateInfo-pStages-06895
If the pipeline requires [fragment shader state](#) but not [pre-rasterization shader state](#), elements of **pStages** **must** not have **stage** set to a shader stage which participates in pre-rasterization
- VUID-VkGraphicsPipelineCreateInfo-pStages-06896
If the pipeline requires [pre-rasterization shader state](#), all elements of **pStages** **must** have a **stage** set to a shader stage which participates in [fragment shader state](#) or [pre-rasterization shader state](#)
- VUID-VkGraphicsPipelineCreateInfo-stage-06897
If the pipeline requires [fragment shader state](#) and/or [pre-rasterization shader state](#), any value of **stage** **must** not be set in more than one element of **pStages**
- VUID-VkGraphicsPipelineCreateInfo-None-08893
The pipeline **must** be created with [pre-rasterization shader state](#)
- VUID-VkGraphicsPipelineCreateInfo-pStages-08894
If **pStages** includes a vertex shader stage, the pipeline **must** be created with [vertex input state](#)
- VUID-VkGraphicsPipelineCreateInfo-pDynamicState-08896
If **pDynamicState->pDynamicStates** includes **VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE**, or if it does not and **pRasterizationState->rasterizerDiscardEnable** is **VK_FALSE**, the pipeline **must** be created with [fragment shader state](#) and [fragment output interface state](#)
- VUID-VkGraphicsPipelineCreateInfo-None-09043

If the format of any color attachment is `VK_FORMAT_E5B9G9R9_UFLOAT_PACK32`, the `colorWriteMask` member of the corresponding element of `pColorBlendState->pAttachments` **must** either include all of `VK_COLOR_COMPONENT_R_BIT`, `VK_COLOR_COMPONENT_G_BIT`, and `VK_COLOR_COMPONENT_B_BIT`, or none of them

Valid Usage (Implicit)

- `VUID-VkGraphicsPipelineCreateInfo-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO`
- `VUID-VkGraphicsPipelineCreateInfo-pNext-pNext`
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkPipelineDiscardRectangleStateCreateInfoEXT`, `VkPipelineFragmentShadingRateStateCreateInfoKHR`, or `VkPipelineOfflineCreateInfo`
- `VUID-VkGraphicsPipelineCreateInfo-sType-unique`
The `sType` value of each structure in the `pNext` chain **must** be unique
- `VUID-VkGraphicsPipelineCreateInfo-pDynamicState-parameter`
If `pDynamicState` is not `NULL`, `pDynamicState` **must** be a valid pointer to a valid `VkPipelineDynamicStateCreateInfo` structure
- `VUID-VkGraphicsPipelineCreateInfo-commonparent`
Each of `basePipelineHandle`, `layout`, and `renderPass` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same `VkDevice`

The `VkPipelineDynamicStateCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineDynamicStateCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkPipelineDynamicStateCreateFlags flags;
    uint32_t           dynamicStateCount;
    const VkDynamicState* pDynamicStates;
} VkPipelineDynamicStateCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `dynamicStateCount` is the number of elements in the `pDynamicStates` array.
- `pDynamicStates` is a pointer to an array of `VkDynamicState` values specifying which pieces of pipeline state will use the values from dynamic state commands rather than from pipeline state creation information.

Valid Usage

- VUID-VkPipelineDynamicStateCreateInfo-pDynamicStates-01442
Each element of `pDynamicStates` **must** be unique

Valid Usage (Implicit)

- VUID-VkPipelineDynamicStateCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO`
- VUID-VkPipelineDynamicStateCreateInfo-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkPipelineDynamicStateCreateInfo-flags-zeroBitmask
`flags` **must** be `0`
- VUID-VkPipelineDynamicStateCreateInfo-pDynamicStates-parameter
If `dynamicStateCount` is not `0`, `pDynamicStates` **must** be a valid pointer to an array of `dynamicStateCount` valid `VkDynamicState` values

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineDynamicStateCreateFlags;
```

`VkPipelineDynamicStateCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

The source of different pieces of dynamic state is specified by the `VkPipelineDynamicStateCreateInfo::pDynamicStates` property of the currently active pipeline, each of whose elements **must** be one of the values:

```
// Provided by VK_VERSION_1_0
typedef enum VkDynamicState {
    VK_DYNAMIC_STATE_VIEWPORT = 0,
    VK_DYNAMIC_STATE_SCISSOR = 1,
    VK_DYNAMIC_STATE_LINE_WIDTH = 2,
    VK_DYNAMIC_STATE_DEPTH_BIAS = 3,
    VK_DYNAMIC_STATE_BLEND_CONSTANTS = 4,
    VK_DYNAMIC_STATE_DEPTH_BOUNDS = 5,
    VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK = 6,
    VK_DYNAMIC_STATE_STENCIL_WRITE_MASK = 7,
    VK_DYNAMIC_STATE_STENCIL_REFERENCE = 8,
    VK_DYNAMIC_STATE_CULL_MODE = 1000267000,
    VK_DYNAMIC_STATE_FRONT_FACE = 1000267001,
    VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY = 1000267002,
    VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT = 1000267003,
    VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT = 1000267004,
    VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE = 1000267005,
```

```

VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE = 1000267006,
VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE = 1000267007,
VK_DYNAMIC_STATE_DEPTH_COMPARE_OP = 1000267008,
VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE = 1000267009,
VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE = 1000267010,
VK_DYNAMIC_STATE_STENCIL_OP = 1000267011,
VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE = 1000377001,
VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE = 1000377002,
VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE = 1000377004,
VK_DYNAMIC_STATE_LINE_STIPPLE = 1000259000,
// Provided by VK_EXT_discard_rectangles
VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT = 1000099000,
// Provided by VK_EXT_discard_rectangles
VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT = 1000099001,
// Provided by VK_EXT_discard_rectangles
VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT = 1000099002,
// Provided by VK_EXT_sample_locations
VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT = 1000143000,
// Provided by VK_KHR_fragment_shading_rate
VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR = 1000226000,
// Provided by VK_EXT_vertex_input_dynamic_state
VK_DYNAMIC_STATE_VERTEX_INPUT_EXT = 1000352000,
// Provided by VK_EXT_extended_dynamic_state2
VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT = 1000377000,
// Provided by VK_EXT_extended_dynamic_state2
VK_DYNAMIC_STATE_LOGIC_OP_EXT = 1000377003,
// Provided by VK_EXT_color_write_enable
VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT = 1000381000,
// Provided by VK_EXT_line_rasterization
VK_DYNAMIC_STATE_LINE_STIPPLE_EXT = VK_DYNAMIC_STATE_LINE_STIPPLE,
// Provided by VK_EXT_extended_dynamic_state
VK_DYNAMIC_STATE_CULL_MODE_EXT = VK_DYNAMIC_STATE_CULL_MODE,
// Provided by VK_EXT_extended_dynamic_state
VK_DYNAMIC_STATE_FRONT_FACE_EXT = VK_DYNAMIC_STATE_FRONT_FACE,
// Provided by VK_EXT_extended_dynamic_state
VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY_EXT = VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY,
// Provided by VK_EXT_extended_dynamic_state
VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT_EXT = VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT,
// Provided by VK_EXT_extended_dynamic_state
VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT_EXT = VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT,
// Provided by VK_EXT_extended_dynamic_state
VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT =
VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE,
// Provided by VK_EXT_extended_dynamic_state
VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE_EXT = VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE,
// Provided by VK_EXT_extended_dynamic_state
VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE_EXT = VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE,
// Provided by VK_EXT_extended_dynamic_state
VK_DYNAMIC_STATE_DEPTH_COMPARE_OP_EXT = VK_DYNAMIC_STATE_DEPTH_COMPARE_OP,
// Provided by VK_EXT_extended_dynamic_state
VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE_EXT =

```

```

VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE,
// Provided by VK_EXT_extended_dynamic_state
VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE_EXT = VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE,
// Provided by VK_EXT_extended_dynamic_state
VK_DYNAMIC_STATE_STENCIL_OP_EXT = VK_DYNAMIC_STATE_STENCIL_OP,
// Provided by VK_EXT_extended_dynamic_state2
VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE_EXT =
VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE,
// Provided by VK_EXT_extended_dynamic_state2
VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE_EXT = VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE,
// Provided by VK_EXT_extended_dynamic_state2
VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE_EXT =
VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE,
// Provided by VK_KHR_line_rasterization
VK_DYNAMIC_STATE_LINE_STIPPLE_KHR = VK_DYNAMIC_STATE_LINE_STIPPLE,
} VkDynamicState;

```

- **VK_DYNAMIC_STATE_VIEWPORT** specifies that the **pViewports** state in **VkPipelineViewportStateCreateInfo** will be ignored and **must** be set dynamically with **vkCmdSetViewport** before any drawing commands. The number of viewports used by a pipeline is still specified by the **viewportCount** member of **VkPipelineViewportStateCreateInfo**.
- **VK_DYNAMIC_STATE_SCISSOR** specifies that the **pScissors** state in **VkPipelineViewportStateCreateInfo** will be ignored and **must** be set dynamically with **vkCmdSetScissor** before any drawing commands. The number of scissor rectangles used by a pipeline is still specified by the **scissorCount** member of **VkPipelineViewportStateCreateInfo**.
- **VK_DYNAMIC_STATE_LINE_WIDTH** specifies that the **lineWidth** state in **VkPipelineRasterizationStateCreateInfo** will be ignored and **must** be set dynamically with **vkCmdSetLineWidth** before any drawing commands that generate line primitives for the rasterizer.
- **VK_DYNAMIC_STATE_DEPTH_BIAS** specifies that the **depthBiasConstantFactor**, **depthBiasClamp** and **depthBiasSlopeFactor** states in **VkPipelineRasterizationStateCreateInfo** will be ignored and **must** be set dynamically with **vkCmdSetDepthBias** before any draws are performed with **depth bias enabled**.
- **VK_DYNAMIC_STATE_BLEND_CONSTANTS** specifies that the **blendConstants** state in **VkPipelineColorBlendStateCreateInfo** will be ignored and **must** be set dynamically with **vkCmdSetBlendConstants** before any draws are performed with a pipeline state with **VkPipelineColorBlendAttachmentState** member **blendEnable** set to **VK_TRUE** and any of the blend functions using a constant blend color.
- **VK_DYNAMIC_STATE_DEPTH_BOUNDS** specifies that the **minDepthBounds** and **maxDepthBounds** states of **VkPipelineDepthStencilStateCreateInfo** will be ignored and **must** be set dynamically with **vkCmdSetDepthBounds** before any draws are performed with a pipeline state with **VkPipelineDepthStencilStateCreateInfo** member **depthBoundsTestEnable** set to **VK_TRUE**.
- **VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK** specifies that the **compareMask** state in **VkPipelineDepthStencilStateCreateInfo** for both **front** and **back** will be ignored and **must** be set dynamically with **vkCmdSetStencilCompareMask** before any draws are performed with a pipeline state with **VkPipelineDepthStencilStateCreateInfo** member **stencilTestEnable** set to

VK_TRUE

- **VK_DYNAMIC_STATE_STENCIL_WRITE_MASK** specifies that the **writeMask** state in **VkPipelineDepthStencilStateCreateInfo** for both **front** and **back** will be ignored and **must** be set dynamically with **vkCmdSetStencilWriteMask** before any draws are performed with a pipeline state with **VkPipelineDepthStencilStateCreateInfo** member **stencilTestEnable** set to **VK_TRUE**
- **VK_DYNAMIC_STATE_STENCIL_REFERENCE** specifies that the **reference** state in **VkPipelineDepthStencilStateCreateInfo** for both **front** and **back** will be ignored and **must** be set dynamically with **vkCmdSetStencilReference** before any draws are performed with a pipeline state with **VkPipelineDepthStencilStateCreateInfo** member **stencilTestEnable** set to **VK_TRUE**
- **VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT** specifies that the **pDiscardRectangles** state in **VkPipelineDiscardRectangleStateCreateInfoEXT** will be ignored and **must** be set dynamically with **vkCmdSetDiscardRectangleEXT** before any draw or clear commands.
- **VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT** specifies that the presence of the **VkPipelineDiscardRectangleStateCreateInfoEXT** structure in the **VkGraphicsPipelineCreateInfo** chain with a **discardRectangleCount** greater than zero does not implicitly enable discard rectangles and they **must** be enabled dynamically with **vkCmdSetDiscardRectangleEnableEXT** before any draw commands. This is available on implementations that support at least **specVersion 2** of the **VK_EXT_discard_rectangles** extension.
- **VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT** specifies that the **discardRectangleMode** state in **VkPipelineDiscardRectangleStateCreateInfoEXT** will be ignored and **must** be set dynamically with **vkCmdSetDiscardRectangleModeEXT** before any draw commands. This is available on implementations that support at least **specVersion 2** of the **VK_EXT_discard_rectangles** extension.
- **VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT** specifies that the **sampleLocationsInfo** state in **VkPipelineSampleLocationsStateCreateInfoEXT** will be ignored and **must** be set dynamically with **vkCmdSetSampleLocationsEXT** before any draw or clear commands. Enabling custom sample locations is still indicated by the **sampleLocationsEnable** member of **VkPipelineSampleLocationsStateCreateInfoEXT**.
- **VK_DYNAMIC_STATE_LINE_STIPPLE** specifies that the **lineStippleFactor** and **lineStipplePattern** state in **VkPipelineRasterizationLineStateCreateInfo** will be ignored and **must** be set dynamically with **vkCmdSetLineStippleKHR** before any draws are performed with a pipeline state with **VkPipelineRasterizationLineStateCreateInfo** member **stippledLineEnable** set to **VK_TRUE**.
- **VK_DYNAMIC_STATE_CULL_MODE** specifies that the **cullMode** state in **VkPipelineRasterizationStateCreateInfo** will be ignored and **must** be set dynamically with **vkCmdSetCullModeEXT** before any drawing commands.
- **VK_DYNAMIC_STATE_FRONT_FACE** specifies that the **frontFace** state in **VkPipelineRasterizationStateCreateInfo** will be ignored and **must** be set dynamically with **vkCmdSetFrontFaceEXT** before any drawing commands.
- **VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY** specifies that the **topology** state in **VkPipelineInputAssemblyStateCreateInfo** only specifies the **topology class**, and the specific topology order and adjacency **must** be set dynamically with **vkCmdSetPrimitiveTopologyEXT** before any drawing commands.
- **VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT** specifies that the **viewportCount** and **pViewports** state in **VkPipelineViewportStateCreateInfo** will be ignored and **must** be set dynamically with

`vkCmdSetViewportWithCountEXT` before any draw call.

- `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` specifies that the `scissorCount` and `pScissors` state in `VkPipelineViewportStateCreateInfo` will be ignored and **must** be set dynamically with `vkCmdSetScissorWithCountEXT` before any draw call.
- `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE` specifies that the `stride` state in `VkVertexInputBindingDescription` will be ignored and **must** be set dynamically with `vkCmdBindVertexBuffers2EXT` before any draw call.
- `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE` specifies that the `depthTestEnable` state in `VkPipelineDepthStencilStateCreateInfo` will be ignored and **must** be set dynamically with `vkCmdSetDepthTestEnableEXT` before any draw call.
- `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` specifies that the `depthWriteEnable` state in `VkPipelineDepthStencilStateCreateInfo` will be ignored and **must** be set dynamically with `vkCmdSetDepthWriteEnableEXT` before any draw call.
- `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP` specifies that the `depthCompareOp` state in `VkPipelineDepthStencilStateCreateInfo` will be ignored and **must** be set dynamically with `vkCmdSetDepthCompareOpEXT` before any draw call.
- `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE` specifies that the `depthBoundsTestEnable` state in `VkPipelineDepthStencilStateCreateInfo` will be ignored and **must** be set dynamically with `vkCmdSetDepthBoundsTestEnableEXT` before any draw call.
- `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE` specifies that the `stencilTestEnable` state in `VkPipelineDepthStencilStateCreateInfo` will be ignored and **must** be set dynamically with `vkCmdSetStencilTestEnableEXT` before any draw call.
- `VK_DYNAMIC_STATE_STENCIL_OP` specifies that the `failOp`, `passOp`, `depthFailOp`, and `compareOp` states in `VkPipelineDepthStencilStateCreateInfo` for both `front` and `back` will be ignored and **must** be set dynamically with `vkCmdSetStencilOpEXT` before any draws are performed with a pipeline state with `VkPipelineDepthStencilStateCreateInfo` member `stencilTestEnable` set to `VK_TRUE`.
- `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT` specifies that the `patchControlPoints` state in `VkPipelineTessellationStateCreateInfo` will be ignored and **must** be set dynamically with `vkCmdSetPatchControlPointsEXT` before any drawing commands.
- `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` specifies that the `rasterizerDiscardEnable` state in `VkPipelineRasterizationStateCreateInfo` will be ignored and **must** be set dynamically with `vkCmdSetRasterizerDiscardEnableEXT` before any drawing commands.
- `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE` specifies that the `depthBiasEnable` state in `VkPipelineRasterizationStateCreateInfo` will be ignored and **must** be set dynamically with `vkCmdSetDepthBiasEnableEXT` before any drawing commands.
- `VK_DYNAMIC_STATE_LOGIC_OP_EXT` specifies that the `logicOp` state in `VkPipelineColorBlendStateCreateInfo` will be ignored and **must** be set dynamically with `vkCmdSetLogicOpEXT` before any drawing commands.
- `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` specifies that the `primitiveRestartEnable` state in `VkPipelineInputAssemblyStateCreateInfo` will be ignored and **must** be set dynamically with `vkCmdSetPrimitiveRestartEnableEXT` before any drawing commands.
- `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` specifies that state in

`VkPipelineFragmentShadingRateStateCreateInfoKHR` will be ignored and **must** be set dynamically with `vkCmdSetFragmentShadingRateKHR` before any drawing commands.

- `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` specifies that the `pVertexInputState` state will be ignored and **must** be set dynamically with `vkCmdSetVertexInputEXT` before any drawing commands
- `VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT` specifies that the `pColorWriteEnables` state in `VkPipelineColorWriteCreateInfoEXT` will be ignored and **must** be set dynamically with `vkCmdSetColorWriteEnableEXT` before any draw call.

10.4.1. Valid Combinations of Stages for Graphics Pipelines

If tessellation shader stages are omitted, the tessellation shading and fixed-function stages of the pipeline are skipped.

If a geometry shader is omitted, the geometry shading stage is skipped.

If a fragment shader is omitted, fragment color outputs have undefined values, and the fragment depth value is determined by `Fragment Operations` state. This **can** be useful for depth-only rendering.

Presence of a shader stage in a pipeline is derived from the `pipeline cache` entry identified by `VkPipelineOfflineCreateInfo::pipelineIdentifier`.

Presence of some of the fixed-function stages in the pipeline is implicitly derived from enabled shaders and provided state. For example, the fixed-function tessellator is always present when the pipeline has valid Tessellation Control and Tessellation Evaluation shaders.

For example:

- Depth/stencil-only rendering in a subpass with no color attachments
 - Active Pipeline Shader Stages
 - Vertex Shader
 - Required: Fixed-Function Pipeline Stages
 - `VkPipelineVertexInputStateCreateInfo`
 - `VkPipelineInputAssemblyStateCreateInfo`
 - `VkPipelineViewportStateCreateInfo`
 - `VkPipelineRasterizationStateCreateInfo`
 - `VkPipelineMultisampleStateCreateInfo`
 - `VkPipelineDepthStencilStateCreateInfo`
- Color-only rendering in a subpass with no depth/stencil attachment
 - Active Pipeline Shader Stages
 - Vertex Shader
 - Fragment Shader
 - Required: Fixed-Function Pipeline Stages

- [VkPipelineVertexInputStateCreateInfo](#)
- [VkPipelineInputAssemblyStateCreateInfo](#)
- [VkPipelineViewportStateCreateInfo](#)
- [VkPipelineRasterizationStateCreateInfo](#)
- [VkPipelineMultisampleStateCreateInfo](#)
- [VkPipelineColorBlendStateCreateInfo](#)
- Rendering pipeline with tessellation and geometry shaders
 - Active Pipeline Shader Stages
 - Vertex Shader
 - Tessellation Control Shader
 - Tessellation Evaluation Shader
 - Geometry Shader
 - Fragment Shader
 - Required: Fixed-Function Pipeline Stages
 - [VkPipelineVertexInputStateCreateInfo](#)
 - [VkPipelineInputAssemblyStateCreateInfo](#)
 - [VkPipelineTessellationStateCreateInfo](#)
 - [VkPipelineViewportStateCreateInfo](#)
 - [VkPipelineRasterizationStateCreateInfo](#)
 - [VkPipelineMultisampleStateCreateInfo](#)
 - [VkPipelineDepthStencilStateCreateInfo](#)
 - [VkPipelineColorBlendStateCreateInfo](#)

10.5. Pipeline Destruction

To destroy a pipeline, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyPipeline(
    VkDevice          device,
    VkPipeline        pipeline,
    const VkAllocationCallbacks* pAllocator);
```

- **device** is the logical device that destroys the pipeline.
- **pipeline** is the handle of the pipeline to destroy.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage

- VUID-vkDestroyPipeline-pipeline-00765
All submitted commands that refer to **pipeline** **must** have completed execution

Valid Usage (Implicit)

- VUID-vkDestroyPipeline-device-parameter
device **must** be a valid **VkDevice** handle
- VUID-vkDestroyPipeline-pipeline-parameter
If **pipeline** is not **VK_NULL_HANDLE**, **pipeline** **must** be a valid **VkPipeline** handle
- VUID-vkDestroyPipeline-pAllocator-null
pAllocator **must** be **NULL**
- VUID-vkDestroyPipeline-pipeline-parent
If **pipeline** is a valid handle, it **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to **pipeline** **must** be externally synchronized

10.6. Pipeline Derivatives

A pipeline derivative is a child pipeline created from a parent pipeline, where the child and parent are expected to have much commonality.

Pipeline derivatives are not supported in Vulkan SC due to the use of read-only offline generated pipeline caches [SCID-8].

10.7. Pipeline Cache

Pipeline cache objects allow the application to load multiple binary pipeline objects generated by an offline cache creation tool into pipeline cache objects. The cache can then be used during pipeline creation to load offline pipeline data.

Pipeline cache objects are represented by **VkPipelineCache** handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkPipelineCache)
```

10.7.1. Creating a Pipeline Cache

To create pipeline cache objects, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreatePipelineCache(
    VkDevice                                device,
    const VkPipelineCacheCreateInfo*        pCreateInfo,
    const VkAllocationCallbacks*           pAllocator,
    VkPipelineCache*                        pPipelineCache);
```

- **device** is the logical device that creates the pipeline cache object.
- **pCreateInfo** is a pointer to a [VkPipelineCacheCreateInfo](#) structure containing initial parameters for the pipeline cache object.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.
- **pPipelineCache** is a pointer to a [VkPipelineCache](#) handle in which the resulting pipeline cache object is returned.

If the pipeline cache data pointed to by [VkPipelineCacheCreateInfo::pInitialData](#) is not compatible with the device, pipeline cache creation will fail and **VK_ERROR_INVALID_PIPELINE_CACHE_DATA** will be returned.

Once created, a pipeline cache **can** be passed to the [vkCreateGraphicsPipelines](#) and [vkCreateComputePipelines](#) commands. The pipeline cache passed into these commands will be queried by the implementation for matching pipelines on pipeline creation. After the cache is created, its contents cannot be updated. The use of the pipeline cache object in these commands is internally synchronized, and the same pipeline cache object **can** be used in multiple threads simultaneously.

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is **VK_TRUE**, [vkCreatePipelineCache](#) **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage

- VUID-vkCreatePipelineCache-pCreateInfo-05045
The values of the **flags**, **initialDataSize**, and **pInitialData** members of the structure pointed to by **pCreateInfo** and the data pointed to by **pCreateInfo->pInitialData** **must** be the same as specified in one of the elements of the **pPipelineCacheCreateInfos** array member of a [VkDeviceObjectReservationCreateInfo](#) structure specified in the **pNext** chain of the [VkDeviceCreateInfo](#) structure **device** was created with
- VUID-vkCreatePipelineCache-device-05068
The number of pipeline caches currently allocated from **device** plus 1 **must** be less than or equal to the total number of pipeline caches requested via [VkDeviceObjectReservationCreateInfo::pipelineCacheRequestCount](#) specified when **device** was created

Valid Usage (Implicit)

- VUID-vkCreatePipelineCache-device-parameter
device must be a valid [VkDevice](#) handle
- VUID-vkCreatePipelineCache-pCreateInfo-parameter
pCreateInfo must be a valid pointer to a valid [VkPipelineCacheCreateInfo](#) structure
- VUID-vkCreatePipelineCache-pAllocator-null
pAllocator must be **NULL**
- VUID-vkCreatePipelineCache-pPipelineCache-parameter
pPipelineCache must be a valid pointer to a [VkPipelineCache](#) handle

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_INVALID_PIPELINE_CACHE_DATA**
- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The [VkPipelineCacheCreateInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineCacheCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkPipelineCacheCreateFlags flags;
    size_t              initialDataSize;
    const void*        pInitialData;
} VkPipelineCacheCreateInfo;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **flags** is a bitmask of [VkPipelineCacheCreateFlagBits](#) specifying the behavior of the pipeline cache.
- **initialDataSize** is the number of bytes in **pInitialData**.
- **pInitialData** is a pointer to pipeline cache data that has been generated offline. If the pipeline cache data is incompatible (as defined below) with the device,

`VK_ERROR_INVALID_PIPELINE_CACHE_DATA` is returned. If `VK_PIPELINE_CACHE_CREATE_USE_APPLICATION_STORAGE_BIT` is set in `flags`, `pInitialData` **may** only be accessed by the implementation during commands using the `VkPipelineCache` object.

NOTE

If `VK_PIPELINE_CACHE_CREATE_USE_APPLICATION_STORAGE_BIT` is set, the memory pointed to by `pInitialData` **can** be modified or released by the application only after any pipeline cache objects created using it have been destroyed.

Valid Usage

- VUID-VkPipelineCacheCreateInfo-flags-05043
`flags` **must** include `VK_PIPELINE_CACHE_CREATE_READ_ONLY_BIT`
- VUID-VkPipelineCacheCreateInfo-flags-05044
`flags` **must** include `VK_PIPELINE_CACHE_CREATE_USE_APPLICATION_STORAGE_BIT`
- VUID-VkPipelineCacheCreateInfo-pInitialData-05139
The pipeline cache data pointed to by `pInitialData` **must** not contain any pipelines with duplicate pipeline identifiers

Valid Usage (Implicit)

- VUID-VkPipelineCacheCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_CACHE_CREATE_INFO`
- VUID-VkPipelineCacheCreateInfo-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkPipelineCacheCreateInfo-flags-parameter
`flags` **must** be a valid combination of `VkPipelineCacheCreateFlagBits` values
- VUID-VkPipelineCacheCreateInfo-pInitialData-parameter
`pInitialData` **must** be a valid pointer to an array of `initialDataSize` bytes
- VUID-VkPipelineCacheCreateInfo-initialDataSize-arraylength
`initialDataSize` **must** be greater than `0`

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineCacheCreateFlags;
```

`VkPipelineCacheCreateFlags` is a bitmask type for setting a mask of zero or more `VkPipelineCacheCreateFlagBits`.

Bits which **can** be set in `VkPipelineCacheCreateInfo::flags`, specifying behavior of the pipeline cache, are:


```
// Provided by VKSC_VERSION_1_0
typedef enum VkPipelineCacheCreateFlagBits {
    // Provided by VKSC_VERSION_1_0
    VK_PIPELINE_CACHE_CREATE_READ_ONLY_BIT = 0x00000002,
    // Provided by VKSC_VERSION_1_0
    VK_PIPELINE_CACHE_CREATE_USE_APPLICATION_STORAGE_BIT = 0x00000004,
} VkPipelineCacheCreateFlagBits;
```

- `VK_PIPELINE_CACHE_CREATE_READ_ONLY_BIT` specifies that the new pipeline cache will be read-only.
- `VK_PIPELINE_CACHE_CREATE_USE_APPLICATION_STORAGE_BIT` specifies that the application will maintain the contents of the memory pointed to by `pInitialData` for the lifetime of the pipeline cache object created, avoiding the need for the implementation to make a copy of the data. The memory pointed to by `pInitialData` **can** be modified or released by the application only after any pipeline cache objects created using it have been destroyed.

10.7.2. Pipeline Cache Header

Applications **must** load data from [offline compiled](#) pipeline caches into pipeline cache objects. The results of pipeline compilations **may** depend on the vendor ID, device ID, driver version, and other details of the target device. To allow detection of pipeline cache data that is incompatible with the device, the pipeline cache data **must** begin with a valid pipeline cache header.

NOTE

Structures described in this section are not part of the Vulkan API and are only used to describe the representation of data elements in pipeline cache data. Accordingly, the valid usage clauses defined for structures defined in this section do not define valid usage conditions for APIs accepting pipeline cache data as input, as providing invalid pipeline cache data as input to any Vulkan API commands will result in the runtime error `VK_ERROR_INVALID_PIPELINE_CACHE_DATA`.

Version one of the pipeline cache header is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineCacheHeaderVersionOne {
    uint32_t          headerSize;
    VkPipelineCacheHeaderVersion headerVersion;
    uint32_t          vendorID;
    uint32_t          deviceID;
    uint8_t           pipelineCacheUUID[VK_UUID_SIZE];
} VkPipelineCacheHeaderVersionOne;
```

- `headerSize` is the length in bytes of the pipeline cache header.
- `headerVersion` is a `VkPipelineCacheHeaderVersion` value specifying the version of the header. A consumer of the pipeline cache **should** use the cache version to interpret the remainder of the cache header. `headerVersion` **must** be written as exactly 4 bytes.
- `vendorID` is the `VkPhysicalDeviceProperties::vendorID` of the implementation.

- `deviceId` is the `VkPhysicalDeviceProperties::deviceId` of the implementation.
- `pipelineCacheUUID` is the `VkPhysicalDeviceProperties::pipelineCacheUUID` of the implementation.

Unlike most structures declared by the Vulkan API, all fields of this structure are written with the least significant byte first, regardless of host byte-order.

The C language specification does not define the packing of structure members. This layout assumes tight structure member packing, with members laid out in the order listed in the structure, and the intended size of the structure is 56 bytes. If a compiler produces code that diverges from that pattern, applications **must** employ another method to set values at the correct offsets.

Valid Usage

- VUID-VkPipelineCacheHeaderVersionOne-headerSize-05075
`headerSize` **must** be 56
- VUID-VkPipelineCacheHeaderVersionOne-headerVersion-05076
`headerVersion` **must** be `VK_PIPELINE_CACHE_HEADER_VERSION_SAFETY_CRITICAL_ONE`
- VUID-VkPipelineCacheHeaderVersionOne-headerSize-08990
`headerSize` **must** not exceed the size of the pipeline cache

Valid Usage (Implicit)

- VUID-VkPipelineCacheHeaderVersionOne-headerVersion-parameter
`headerVersion` **must** be a valid `VkPipelineCacheHeaderVersion` value

Possible values of the `headerVersion` value of the pipeline cache header are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPipelineCacheHeaderVersion {
    VK_PIPELINE_CACHE_HEADER_VERSION_ONE = 1,
    // Provided by VKSC_VERSION_1_0
    VK_PIPELINE_CACHE_HEADER_VERSION_SAFETY_CRITICAL_ONE = 1000298001,
} VkPipelineCacheHeaderVersion;
```

- `VK_PIPELINE_CACHE_HEADER_VERSION_ONE` specifies version one of the pipeline cache, described by `VkPipelineCacheHeaderVersionOne`.
- `VK_PIPELINE_CACHE_HEADER_VERSION_SAFETY_CRITICAL_ONE` specifies version one of the pipeline cache for Vulkan SC, described by `VkPipelineCacheHeaderVersionSafetyCriticalOne`.

Version one of the pipeline cache header for Vulkan SC is defined as:

```
// Provided by VKSC_VERSION_1_0
typedef struct VkPipelineCacheHeaderVersionSafetyCriticalOne {
    VkPipelineCacheHeaderVersionOne    headerVersionOne;
    VkPipelineCacheValidationVersion    validationVersion;
    uint32_t                           implementationData;
    uint32_t                           pipelineIndexCount;
    uint32_t                           pipelineIndexStride;
    uint64_t                           pipelineIndexOffset;
} VkPipelineCacheHeaderVersionSafetyCriticalOne;
```

- `headerVersionOne` is a [VkPipelineCacheHeaderVersionOne](#) structure.
- `validationVersion` is a [VkPipelineCacheValidationVersion](#) enum value specifying the version of any validation information that is included in this pipeline cache. `validationVersion` **must** be written as exactly 4 bytes.
- `implementationData` is 4 bytes of padding to ensure structure members are consistently aligned on all platforms. The contents of this field **may** be used for implementation-specific information.
- `pipelineIndexCount` is the number of entries contained in the pipeline cache index.
- `pipelineIndexStride` is the number of bytes between consecutive pipeline cache index entries.
- `pipelineIndexOffset` is the offset in bytes from the beginning of the pipeline cache header to the pipeline cache index.

The [pipeline cache index](#) consists of `pipelineIndexCount` [VkPipelineCacheSafetyCriticalIndexEntry](#) structures containing an index of all the pipelines in this cache. The pipeline cache index is located starting at `pipelineIndexOffset` bytes into the cache and the location of pipeline `i` is calculated as: `pipelineIndexOffset + i × pipelineIndexStride`. The [VkPipelineCacheSafetyCriticalIndexEntry](#) structures **may** not be tightly packed, enabling additional implementation-specific data to be stored with each entry, or for future extensibility.

NOTE

Because the pipeline cache index is keyed by pipeline identifier, applications and offline compilers must ensure that there are no pipelines with identical pipeline identifiers in the same pipeline cache.

Unlike most structures declared by the Vulkan API, all fields of this structure are written with the least significant byte first, regardless of host byte-order.

The C language specification does not define the packing of structure members. This layout assumes tight structure member packing, with members laid out in the order listed in the structure, and the intended size of the structure is 56 bytes. If a compiler produces code that diverges from that pattern, applications **must** employ another method to set values at the correct offsets.

Valid Usage

- VUID-VkPipelineCacheHeaderVersionSafetyCriticalOne-validationVersion-05077
`validationVersion` **must** be `VK_PIPELINE_CACHE_VALIDATION_VERSION_SAFETY_CRITICAL_ONE`

- VUID-VkPipelineCacheHeaderVersionSafetyCriticalOne-pipelineIndexStride-05078 `pipelineIndexStride` **must** be greater than or equal to 56 (the size of the `VkPipelineCacheSafetyCriticalIndexEntry` structure)
- VUID-VkPipelineCacheHeaderVersionSafetyCriticalOne-pipelineIndexOffset-05079 `pipelineIndexOffset + pipelineIndexCount × pipelineIndexStride` **must** not exceed the size of the pipeline cache

Valid Usage (Implicit)

- VUID-VkPipelineCacheHeaderVersionSafetyCriticalOne-headerVersionOne-parameter `headerVersionOne` **must** be a valid `VkPipelineCacheHeaderVersionOne` structure
- VUID-VkPipelineCacheHeaderVersionSafetyCriticalOne-validationVersion-parameter `validationVersion` **must** be a valid `VkPipelineCacheValidationVersion` value

The `VkPipelineCacheValidationVersion` enumeration determines the contents of the pipeline cache validation information. Possible values are:

```
// Provided by VKSC_VERSION_1_0
typedef enum VkPipelineCacheValidationVersion {
    VK_PIPELINE_CACHE_VALIDATION_VERSION_SAFETY_CRITICAL_ONE = 1,
} VkPipelineCacheValidationVersion;
```

- `VK_PIPELINE_CACHE_VALIDATION_VERSION_SAFETY_CRITICAL_ONE` specifies version one of the pipeline cache validation information for Vulkan SC.

Each pipeline cache index entry consists of a `VkPipelineCacheSafetyCriticalIndexEntry` structure:

```
// Provided by VKSC_VERSION_1_0
typedef struct VkPipelineCacheSafetyCriticalIndexEntry {
    uint8_t      pipelineIdentifier[VK_UUID_SIZE];
    uint64_t     pipelineMemorySize;
    uint64_t     jsonSize;
    uint64_t     jsonOffset;
    uint32_t     stageIndexCount;
    uint32_t     stageIndexStride;
    uint64_t     stageIndexOffset;
} VkPipelineCacheSafetyCriticalIndexEntry;
```

- `pipelineIdentifier` is the `pipeline identifier` indicating which pipeline the information is associated with.
- `pipelineMemorySize` is the number of bytes of pipeline memory required for this pipeline. This is the minimum value that **can** be successfully used for `VkPipelineOfflineCreateInfo::poolEntrySize` when this pipeline is used.
- `jsonSize` is the size in bytes of the pipeline JSON data representing the pipeline state for this

pipeline. This value **may** be zero, indicating the JSON data is not present in the pipeline cache for this pipeline.

- **jsonOffset** is the offset in bytes from the beginning of the pipeline cache header to the pipeline JSON data for this pipeline. This value **must** be zero if the JSON data is not present in the pipeline cache for this pipeline.
- **stageIndexCount** is the number of entries in the pipeline cache stage validation index for this pipeline. This value **may** be zero, indicating that no stage validation information is present in the pipeline cache for this pipeline.
- **stageIndexStride** is the number of bytes between consecutive stage validation index entries.
- **stageIndexOffset** is the offset in bytes from the beginning of the pipeline cache header to the **stage validation index** for this pipeline. This value **must** be zero if no stage validation information is present for this pipeline.

The JSON data and the stage validation index are **optionally** included in the pipeline cache index entry. They are only intended to be used for validation and debugging. If present they **must** include both the JSON data and the corresponding SPIR-V modules that were used by the offline compiler to compile the pipeline cache entry.

The data at **jsonOffset** consists of a byte stream of **jsonSize** bytes of UTF-8 encoded JSON that was used by the **offline pipeline compiler** to create this pipeline cache entry.

The **stage validation index** consists of **stageIndexCount** **VkPipelineCacheStageValidationIndexEntry** structures which provide the SPIR-V modules used by this pipeline and these are provided in the same order as provided to the **VkPipelineShaderStageCreateInfo** structure(s) in the **Vk*PipelineCreateInfo** structure for this pipeline. The stage validation index is located at **stageIndexOffset** bytes into the cache and the location of stage **i** is calculated as: **stageIndexOffset + i × stageIndexStride**. The **VkPipelineCacheStageValidationIndexEntry** structures **may** not be tightly packed, enabling additional implementation-specific data to be stored with each entry, or for future extensibility.

Unlike most structures declared by the Vulkan API, all fields of this structure are written with the least significant byte first, regardless of host byte-order.

The C language specification does not define the packing of structure members. This layout assumes tight structure member packing, with members laid out in the order listed in the structure, and the intended size of the structure is 56 bytes. If a compiler produces code that diverges from that pattern, applications **must** employ another method to set values at the correct offsets.

Valid Usage

- VUID-VkPipelineCacheSafetyCriticalIndexEntry-jsonSize-05080
If **jsonSize** is 0, **jsonOffset** **must** be 0
- VUID-VkPipelineCacheSafetyCriticalIndexEntry-jsonSize-05081
If **jsonSize** is 0, **stageIndexCount** **must** be 0
- VUID-VkPipelineCacheSafetyCriticalIndexEntry-jsonSize-08991
If **jsonSize** is not 0, **jsonOffset + jsonSize** **must** not exceed the size of the pipeline cache

- VUID-VkPipelineCacheSafetyCriticalIndexEntry-stageIndexCount-05082
If `stageIndexCount` is 0, `stageIndexOffset` and `stageIndexStride` **must** be 0
- VUID-VkPipelineCacheSafetyCriticalIndexEntry-stageIndexCount-05083
If `stageIndexCount` is not 0, `stageIndexStride` **must** be greater than or equal to 16 (the size of the `VkPipelineCacheStageValidationIndexEntry` structure)
- VUID-VkPipelineCacheSafetyCriticalIndexEntry-stageIndexCount-05084
If `stageIndexCount` is not 0, `stageIndexOffset` + `stageIndexCount` × `stageIndexStride` **must** not exceed the size of the pipeline cache

Each pipeline cache stage validation index entry consists of a `VkPipelineCacheStageValidationIndexEntry` structure:

```
// Provided by VKSC_VERSION_1_0
typedef struct VkPipelineCacheStageValidationIndexEntry {
    uint64_t    codeSize;
    uint64_t    codeOffset;
} VkPipelineCacheStageValidationIndexEntry;
```

- `codeSize` is the size in bytes of the SPIR-V module for this pipeline stage.
- `codeOffset` is the offset in bytes from the beginning of the pipeline cache header to the SPIR-V module for this pipeline stage.

The data at `codeOffset` consists of `codeSize` bytes of SPIR-V module as described in [Appendix A](#) that was used by the [offline pipeline compiler](#) for this shader stage when creating this pipeline cache entry.

Unlike most structures declared by the Vulkan API, all fields of this structure are written with the least significant byte first, regardless of host byte-order.

The C language specification does not define the packing of structure members. This layout assumes tight structure member packing, with members laid out in the order listed in the structure, and the intended size of the structure is 16 bytes. If a compiler produces code that diverges from that pattern, applications **must** employ another method to set values at the correct offsets.

Valid Usage

- VUID-VkPipelineCacheStageValidationIndexEntry-codeSize-05085
`codeSize` **must** be greater than 0
- VUID-VkPipelineCacheStageValidationIndexEntry-codeSize-05086
`codeSize` **must** be a multiple of 4
- VUID-VkPipelineCacheStageValidationIndexEntry-codeOffset-05087
`codeOffset` + `codeSize` **must** not exceed the size of the pipeline cache

10.7.3. Destroying a Pipeline Cache

To destroy a pipeline cache, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyPipelineCache(
    VkDevice          device,
    VkPipelineCache   pipelineCache,
    const VkAllocationCallbacks* pAllocator);
```

- **device** is the logical device that destroys the pipeline cache object.
- **pipelineCache** is the handle of the pipeline cache to destroy.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage (Implicit)

- VUID-vkDestroyPipelineCache-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkDestroyPipelineCache-pipelineCache-parameter
If **pipelineCache** is not [VK_NULL_HANDLE](#), **pipelineCache** **must** be a valid [VkPipelineCache](#) handle
- VUID-vkDestroyPipelineCache-pAllocator-null
pAllocator **must** be [NULL](#)
- VUID-vkDestroyPipelineCache-pipelineCache-parent
If **pipelineCache** is a valid handle, it **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to **pipelineCache** **must** be externally synchronized

10.8. Offline Pipeline Compilation

In Vulkan SC, the pipeline compilation process occurs offline [\[SCID-8\]](#).

The [SPIR-V shader module](#) and pipeline state are supplied to an offline pipeline cache compiler which creates a pipeline cache entry for the pipeline. The set of pipeline cache entries are combined offline into one or more [pipeline caches](#). At application runtime, the offline generated pipeline cache is provided to device creation as part of the [VkDeviceObjectReservationCreateInfo](#) structure and then loaded into a [VkPipelineCache](#) object by the application. The device, pipeline, and pipeline cache creation functions **can** extract implementation-specific information from the pipeline cache. The specific pipeline to be loaded from the cache is specified at pipeline creation time using a [pipeline identifier](#). The pipeline state that is provided at runtime to pipeline creation

must match the state that was specified to the offline pipeline cache compiler when the pipeline cache entry was created offline (with the exception of the [VkPipelineShaderStageCreateInfo](#) structure).

In order to assist with the specification of pipeline state for the offline pipeline cache compiler, Khronos has defined a *pipeline JSON schema* to represent the pipeline state required to compile a SPIR-V module to device-specific machine code and a set of utilities to help with reading and writing of the JSON files. See <https://github.com/KhronosGroup/VulkanSC-Docs/wiki/JSON-schema> for more information.

10.9. Pipeline Memory Reservation

Pipeline memory is allocated from a pool that is reserved at device creation time. The offline pipeline cache compiler writes the pipeline memory size requirements for each pipeline into the pipeline's [VkPipelineCacheSafetyCriticalIndexEntry::pipelineMemorySize](#) entry in the [pipeline cache index](#). The offline pipeline cache compiler **may** also report it separately. The elements of [VkDeviceObjectReservationCreateInfo::pPipelinePoolSizes](#) are requests for [poolEntryCount](#) pool entries each of pool size [poolEntrySize](#), and any pipeline with a [VkPipelineCacheSafetyCriticalIndexEntry::pipelineMemorySize](#) less than or equal to [VkPipelineOfflineCreateInfo::poolEntrySize](#) **can** be placed in one of those pool entries. The application **should** request a set of pool sizes that best suits its anticipated worst-case usage.

On implementations where [VkPhysicalDeviceVulkanSC10Properties::recyclePipelineMemory](#) is [VK_FALSE](#), the memory for the pipeline pool is not recycled when a pipeline is destroyed, and once an entry has been used it **cannot** be reused. On implementations where [VkPhysicalDeviceVulkanSC10Properties::recyclePipelineMemory](#) is [VK_TRUE](#), the memory for the pipeline pool is recycled when a pipeline is destroyed, and the entry it was using becomes available to be reused.

10.10. Pipeline Identifier

A *pipeline identifier* is an identifier that can be used to identify a specific pipeline independently from the pipeline description, shader stages and any relevant fixed-function stages, that were used to create the pipeline object.

The [VkPipelineOfflineCreateInfo](#) structure allows an identifier to be specified for the pipeline at pipeline creation via the [pNext](#) field of the [VkGraphicsPipelineCreateInfo](#), and [VkComputePipelineCreateInfo](#) structures. If a [VkPipelineOfflineCreateInfo](#) structure is not included in the [pNext](#) chain then pipeline creation will fail and [VK_ERROR_NO_PIPELINE_MATCH](#) will be returned by the corresponding [vkCreate*Pipelines](#) command.

The identifier **must** be used by the implementation to match against the existing content of the pipeline cache at pipeline creation. This is required for Vulkan SC where pipelines are generated offline and there is no shader code in the pipeline cache to match at runtime.

NOTE

The identifier values must be specified or generated during the offline pipeline cache generation and embedded in to the pipeline cache blob.

The `VkPipelineOfflineCreateInfo` structure is defined as:

```
// Provided by VKSC_VERSION_1_0
typedef struct VkPipelineOfflineCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    uint8_t              pipelineIdentifier[VK_UUID_SIZE];
    VkPipelineMatchControl matchControl;
    VkDeviceSize          poolEntrySize;
} VkPipelineOfflineCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pipelineIdentifier` is an array of `VK_UUID_SIZE` `uint8_t` values representing an identifier for the pipeline.
- `matchControl` is a `VkPipelineMatchControl` value specifying the type of identifier being used and how the match should be performed.
- `poolEntrySize` is the size of the entry in pipeline memory to use for this pipeline. It **must** be a size that was requested via `VkPipelinePoolSize` when the device was created.

If a match in the pipeline cache is not found then `VK_ERROR_NO_PIPELINE_MATCH` will be returned to the application.

If `poolEntrySize` is too small for the pipeline, or the number of entries for the requested pool size exceeds the reserved count for that pool size, pipeline creation will fail and `VK_ERROR_OUT_OF_POOL_MEMORY` will be returned by the corresponding `vkCreate*Pipelines` command.

Valid Usage

- VUID-VkPipelineOfflineCreateInfo-poolEntrySize-05028
`poolEntrySize` **must** be one of the sizes requested via `VkPipelinePoolSize` when the device was created
- VUID-VkPipelineOfflineCreateInfo-recyclePipelineMemory-05029
If `VkPhysicalDeviceVulkanSC10Properties::recyclePipelineMemory` is `VK_TRUE`, the number of currently existing pipelines created with this same value of `poolEntrySize` plus 1 **must** be less than or equal to the sum of the `VkPipelinePoolSize::poolEntryCount` values with the same value of `poolEntrySize`
- VUID-VkPipelineOfflineCreateInfo-recyclePipelineMemory-05030
If `VkPhysicalDeviceVulkanSC10Properties::recyclePipelineMemory` is `VK_FALSE`, the total number of pipelines ever created with this same value of `poolEntrySize` plus 1 **must** be less than or equal to the sum of the `VkPipelinePoolSize::poolEntryCount` values with the same value of `poolEntrySize`

Valid Usage (Implicit)

- VUID-VkPipelineOfflineCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_OFFLINE_CREATE_INFO`
- VUID-VkPipelineOfflineCreateInfo-matchControl-parameter
`matchControl` **must** be a valid `VkPipelineMatchControl` value

Possible values of the `matchControl` member of `VkPipelineOfflineCreateInfo`

```
// Provided by VKSC_VERSION_1_0
typedef enum VkPipelineMatchControl {
    VK_PIPELINE_MATCH_CONTROL_APPLICATION_UUID_EXACT_MATCH = 0,
} VkPipelineMatchControl;
```

are:

- `VK_PIPELINE_MATCH_CONTROL_APPLICATION_UUID_EXACT_MATCH` specifies that the identifier is a UUID generated by the application and the identifiers **must** be an exact match.

10.11. Specialization Constants

Specialization constants are a mechanism whereby constants in a SPIR-V module **can** have their constant value specified at the time the `VkPipeline` is compiled offline. This allows a SPIR-V module to have constants that **can** be modified at compilation time rather than in the SPIR-V source. The `pSpecializationInfo` parameters are not used at runtime and **should** be ignored by the implementation. If provided, the application **must** set the `pSpecializationInfo` parameters to the values that were specified for the offline compilation of this pipeline.

NOTE

Specialization constants are useful to allow a compute shader to have its local workgroup size changed at pipeline compilation time, for example.

Each `VkPipelineShaderStageCreateInfo` structure contains a `pSpecializationInfo` member, which **can** be `NULL` to indicate no specialization constants, or point to a `VkSpecializationInfo` structure.

The `VkSpecializationInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSpecializationInfo {
    uint32_t                mapEntryCount;
    const VkSpecializationMapEntry* pMapEntries;
    size_t                  dataSize;
    const void*              pData;
} VkSpecializationInfo;
```

- `mapEntryCount` is the number of entries in the `pMapEntries` array.

- `pMapEntries` is a pointer to an array of `VkSpecializationMapEntry` structures, which map constant IDs to offsets in `pData`.
- `dataSize` is the byte size of the `pData` buffer.
- `pData` contains the actual constant values to specialize with.

Valid Usage

- VUID-VkSpecializationInfo-offset-00773
The `offset` member of each element of `pMapEntries` **must** be less than `dataSize`
- VUID-VkSpecializationInfo-pMapEntries-00774
The `size` member of each element of `pMapEntries` **must** be less than or equal to `dataSize` minus `offset`
- VUID-VkSpecializationInfo-constantID-04911
The `constantID` value of each element of `pMapEntries` **must** be unique within `pMapEntries`

Valid Usage (Implicit)

- VUID-VkSpecializationInfo-pMapEntries-parameter
If `mapEntryCount` is not 0, `pMapEntries` **must** be a valid pointer to an array of `mapEntryCount` valid `VkSpecializationMapEntry` structures
- VUID-VkSpecializationInfo-pData-parameter
If `dataSize` is not 0, `pData` **must** be a valid pointer to an array of `dataSize` bytes

The `VkSpecializationMapEntry` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSpecializationMapEntry {
    uint32_t    constantID;
    uint32_t    offset;
    size_t      size;
} VkSpecializationMapEntry;
```

- `constantID` is the ID of the specialization constant in SPIR-V.
- `offset` is the byte offset of the specialization constant value within the supplied data buffer.
- `size` is the byte size of the specialization constant value within the supplied data buffer.

If a `constantID` value is not a specialization constant ID used in the shader, that map entry does not affect the behavior of the pipeline.

Valid Usage

- VUID-VkSpecializationMapEntry-constantID-00776

For a `constantID` specialization constant declared in a shader, `size` **must** match the byte size of the `constantID`. If the specialization constant is of type `boolean`, `size` **must** be the byte size of `VkBool32`

In human readable SPIR-V:

```
OpDecorate %x SpecId 13 ; decorate .x component of WorkgroupSize with ID 13
OpDecorate %y SpecId 42 ; decorate .y component of WorkgroupSize with ID 42
OpDecorate %z SpecId 3 ; decorate .z component of WorkgroupSize with ID 3
OpDecorate %wgsz BuiltIn WorkgroupSize ; decorate WorkgroupSize onto constant
%i32 = OpTypeInt 32 0 ; declare an unsigned 32-bit type
%uvec3 = OpTypeVector %i32 3 ; declare a 3 element vector type of unsigned 32-bit
%x = OpSpecConstant %i32 1 ; declare the .x component of WorkgroupSize
%y = OpSpecConstant %i32 1 ; declare the .y component of WorkgroupSize
%z = OpSpecConstant %i32 1 ; declare the .z component of WorkgroupSize
%wgsz = OpSpecConstantComposite %uvec3 %x %y %z ; declare WorkgroupSize
```

From the above we have three specialization constants, one for each of the x, y and z elements of the WorkgroupSize vector.

Now to specialize the above via the specialization constants mechanism:

```
const VkSpecializationMapEntry entries[] =
{
    {
        .constantID = 13,
        .offset = 0 * sizeof(uint32_t),
        .size = sizeof(uint32_t)
    },
    {
        .constantID = 42,
        .offset = 1 * sizeof(uint32_t),
        .size = sizeof(uint32_t)
    },
    {
        .constantID = 3,
        .offset = 2 * sizeof(uint32_t),
        .size = sizeof(uint32_t)
    }
};

const uint32_t data[] = { 16, 8, 4 }; // our workgroup size is 16x8x4

const VkSpecializationInfo info =
{
    .mapEntryCount = 3,
    .pMapEntries = entries,
    .dataSize = 3 * sizeof(uint32_t),
```

```

        .pData = data,
    };

```

Then when calling `vkCreateComputePipelines`, and passing the `VkSpecializationInfo` we defined as the `pSpecializationInfo` parameter of `VkPipelineShaderStageCreateInfo`, we will create a compute pipeline with the runtime specified local workgroup size.

Another example would be that an application has a SPIR-V module that has some platform-dependent constants they wish to use.

In human readable SPIR-V:

```

OpDecorate %1 SpecId 0 ; decorate our signed 32-bit integer constant
OpDecorate %2 SpecId 12 ; decorate our 32-bit floating-point constant
%i32 = OpTypeInt 32 1 ; declare a signed 32-bit type
%float = OpTypeFloat 32 ; declare a 32-bit floating-point type
%1 = OpSpecConstant %i32 -1 ; some signed 32-bit integer constant
%2 = OpSpecConstant %float 0.5 ; some 32-bit floating-point constant

```

From the above we have two specialization constants, one is a signed 32-bit integer and the second is a 32-bit floating-point value.

Now to specialize the above via the specialization constants mechanism:

```

struct SpecializationData {
    int32_t data0;
    float data1;
};

const VkSpecializationMapEntry entries[] =
{
    {
        .constantID = 0,
        .offset = offsetof(SpecializationData, data0),
        .size = sizeof(SpecializationData::data0)
    },
    {
        .constantID = 12,
        .offset = offsetof(SpecializationData, data1),
        .size = sizeof(SpecializationData::data1)
    }
};

SpecializationData data;
data.data0 = -42; // set the data for the 32-bit integer
data.data1 = 42.0f; // set the data for the 32-bit floating-point

const VkSpecializationInfo info =
{

```

```
.mapEntryCount = 2,
.pMapEntries = entries,
.dataSize = sizeof(data),
.pdata = &data,
};
```

It is legal for a SPIR-V module with specializations to be compiled into a pipeline where no specialization information was provided. SPIR-V specialization constants contain default values such that if a specialization is not provided, the default value will be used. In the examples above, it would be valid for an application to only specialize some of the specialization constants within the SPIR-V module, and let the other constants use their default values encoded within the `OpSpecConstant` declarations.

10.12. Pipeline Binding

Once a pipeline has been created, it **can** be bound to the command buffer using the command:

```
// Provided by VK_VERSION_1_0
void vkCmdBindPipeline(
    VkCommandBuffer          commandBuffer,
    VkPipelineBindPoint      pipelineBindPoint,
    VkPipeline                pipeline);
```

- `commandBuffer` is the command buffer that the pipeline will be bound to.
- `pipelineBindPoint` is a `VkPipelineBindPoint` value specifying to which bind point the pipeline is bound. Binding one does not disturb the others.
- `pipeline` is the pipeline to be bound.

Once bound, a pipeline binding affects subsequent commands that interact with the given pipeline type in the command buffer until a different pipeline of the same type is bound to the bind point. Commands that do not interact with the `given pipeline` type **must** not be affected by the pipeline state.

Valid Usage

- VUID-vkCmdBindPipeline-pipelineBindPoint-00777
If `pipelineBindPoint` is `VK_PIPELINE_BIND_POINT_COMPUTE`, the `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdBindPipeline-pipelineBindPoint-00778
If `pipelineBindPoint` is `VK_PIPELINE_BIND_POINT_GRAPHICS`, the `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdBindPipeline-pipelineBindPoint-00779
If `pipelineBindPoint` is `VK_PIPELINE_BIND_POINT_COMPUTE`, `pipeline` **must** be a compute pipeline
- VUID-vkCmdBindPipeline-pipelineBindPoint-00780

If `pipelineBindPoint` is `VK_PIPELINE_BIND_POINT_GRAPHICS`, `pipeline` **must** be a graphics pipeline

- VUID-vkCmdBindPipeline-pipeline-00781

If the `variableMultisampleRate` feature is not supported, `pipeline` is a graphics pipeline, the current subpass **uses no attachments**, and this is not the first call to this function with a graphics pipeline after transitioning to the current subpass, then the sample count specified by this pipeline **must** match that set in the previous pipeline

- VUID-vkCmdBindPipeline-variableSampleLocations-01525

If `VkPhysicalDeviceSampleLocationsPropertiesEXT::variableSampleLocations` is `VK_FALSE`, and `pipeline` is a graphics pipeline created with a `renderPass` that is not `VK_NULL_HANDLE` and with a `VkPipelineSampleLocationsStateCreateInfoEXT` structure having its `sampleLocationsEnable` member set to `VK_TRUE` but without `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` enabled then the current render pass instance **must** have been begun by specifying a `VkRenderPassSampleLocationsBeginInfoEXT` structure whose `pPostSubpassSampleLocations` member contains an element with a `subpassIndex` matching the current subpass index and the `sampleLocationsInfo` member of that element **must** match the `sampleLocationsInfo` specified in `VkPipelineSampleLocationsStateCreateInfoEXT` when the pipeline was created

- VUID-vkCmdBindPipeline-commandBuffer-04809

If `commandBuffer` is a secondary command buffer with `VkCommandBufferInheritanceViewportScissorInfoNV::viewportScissor2D` enabled and `pipelineBindPoint` is `VK_PIPELINE_BIND_POINT_GRAPHICS` and `pipeline` was created with `VkPipelineDiscardRectangleStateCreateInfoEXT` structure and its `discardRectangleCount` member is not `0`, or the pipeline was created with `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT` enabled, then the pipeline **must** have been created with `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT` enabled

Valid Usage (Implicit)

- VUID-vkCmdBindPipeline-commandBuffer-parameter

`commandBuffer` **must** be a valid `VkCommandBuffer` handle

- VUID-vkCmdBindPipeline-pipelineBindPoint-parameter

`pipelineBindPoint` **must** be a valid `VkPipelineBindPoint` value

- VUID-vkCmdBindPipeline-pipeline-parameter

`pipeline` **must** be a valid `VkPipeline` handle

- VUID-vkCmdBindPipeline-commandBuffer-recording

`commandBuffer` **must** be in the `recording state`

- VUID-vkCmdBindPipeline-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` operations

- VUID-vkCmdBindPipeline-commonparent

Both of `commandBuffer`, and `pipeline` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT	State

Possible values of `vkCmdBindPipeline::pipelineBindPoint`, specifying the bind point of a pipeline object, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPipelineBindPoint {
    VK_PIPELINE_BIND_POINT_GRAPHICS = 0,
    VK_PIPELINE_BIND_POINT_COMPUTE = 1,
} VkPipelineBindPoint;
```

- `VK_PIPELINE_BIND_POINT_COMPUTE` specifies binding as a compute pipeline.
- `VK_PIPELINE_BIND_POINT_GRAPHICS` specifies binding as a graphics pipeline.

10.13. Dynamic State

When a pipeline object is bound, any pipeline object state that is not specified as dynamic is applied to the command buffer state. Pipeline object state that is specified as dynamic is not applied to the command buffer state at this time.

Instead, dynamic state **can** be modified at any time and persists for the lifetime of the command buffer, or until modified by another dynamic state setting command, or made invalid by binding a pipeline in which that state is statically specified.

When a pipeline object is bound, the following applies to each state parameter:

- If the state is not specified as dynamic in the new pipeline object, then that command buffer state is overwritten by the state in the new pipeline object. Before any draw or dispatch call with this pipeline there **must** not have been any calls to any of the corresponding dynamic state setting commands after this pipeline was bound.

- If the state is specified as dynamic in the new pipeline object, then that command buffer state is not disturbed. Before any draw or dispatch call with this pipeline there **must** have been at least one call to each of the corresponding dynamic state setting commands. The state-setting commands **must** be recorded after command buffer recording was begun, or after the last command binding a pipeline object with that state specified as static, whichever was the latter.
- If the state is not included (corresponding pointer in [VkGraphicsPipelineCreateInfo](#) was **NULL** or was ignored) in the new pipeline object, then that command buffer state is not disturbed.

Dynamic state that does not affect the result of operations **can** be left undefined.

NOTE

For example, if blending is disabled by the pipeline object state then the dynamic color blend constants do not need to be specified in the command buffer, even if this state is specified as dynamic in the pipeline object.

NOTE

Applications running on Vulkan implementations advertising a [VkPhysicalDeviceDriverProperties::conformanceVersion](#) less than 1.3.8.0 should be aware that rebinding the bound pipeline object may not reapply static state.

Chapter 11. Memory Allocation

Vulkan memory is broken up into two categories, *host memory* and *device memory*.

11.1. Host Memory

Host memory is memory needed by the Vulkan implementation for non-device-visible storage.

NOTE This memory **may** be used to store the implementation's representation and state of Vulkan objects.

The Vulkan SC implementation will perform its own host memory allocations. Support for application-provided memory allocation, as supported in Base Vulkan, has been removed in Vulkan SC.

`VkAllocationCallbacks` is not supported and pointers to this type **must** be `NULL` [SCID-2], [SCID-8].

```
// Provided by VK_VERSION_1_0
typedef struct VkAllocationCallbacks {
    void*                                pUserData;
    PFN_vkAllocationFunction             pfnAllocation;
    PFN_vkReallocationFunction           pfnReallocation;
    PFN_vkFreeFunction                   pfnFree;
    PFN_vkInternalAllocationNotification pfnInternalAllocation;
    PFN_vkInternalFreeNotification       pfnInternalFree;
} VkAllocationCallbacks;
```

11.2. Device Memory

Device memory is memory that is visible to the device—for example the contents of the image or buffer objects, which **can** be natively used by the device.

11.2.1. Device Memory Properties

Memory properties of a physical device describe the memory heaps and memory types available.

To query memory properties, call:

WARNING This functionality is superseded by [Vulkan Version 1.1](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
void vkGetPhysicalDeviceMemoryProperties(
    VkPhysicalDevice             physicalDevice,
    VkPhysicalDeviceMemoryProperties* pMemoryProperties);
```

- `physicalDevice` is the handle to the device to query.
- `pMemoryProperties` is a pointer to a `VkPhysicalDeviceMemoryProperties` structure in which the properties are returned.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceMemoryProperties-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceMemoryProperties-pMemoryProperties-parameter `pMemoryProperties` **must** be a valid pointer to a `VkPhysicalDeviceMemoryProperties` structure

The `VkPhysicalDeviceMemoryProperties` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPhysicalDeviceMemoryProperties {
    uint32_t      memoryTypeCount;
    VkMemoryType  memoryTypes[VK_MAX_MEMORY_TYPES];
    uint32_t      memoryHeapCount;
    VkMemoryHeap  memoryHeaps[VK_MAX_MEMORY_HEAPS];
} VkPhysicalDeviceMemoryProperties;
```

- `memoryTypeCount` is the number of valid elements in the `memoryTypes` array.
- `memoryTypes` is an array of `VK_MAX_MEMORY_TYPES` `VkMemoryType` structures describing the *memory types* that **can** be used to access memory allocated from the heaps specified by `memoryHeaps`.
- `memoryHeapCount` is the number of valid elements in the `memoryHeaps` array.
- `memoryHeaps` is an array of `VK_MAX_MEMORY_HEAPS` `VkMemoryHeap` structures describing the *memory heaps* from which memory **can** be allocated.

The `VkPhysicalDeviceMemoryProperties` structure describes a number of *memory heaps* as well as a number of *memory types* that **can** be used to access memory allocated in those heaps. Each heap describes a memory resource of a particular size, and each memory type describes a set of memory properties (e.g. host cached vs. uncached) that **can** be used with a given memory heap. Allocations using a particular memory type will consume resources from the heap indicated by that memory type's heap index. More than one memory type **may** share each heap, and the heaps and memory types provide a mechanism to advertise an accurate size of the physical memory resources while allowing the memory to be used with a variety of different properties.

The number of memory heaps is given by `memoryHeapCount` and is less than or equal to `VK_MAX_MEMORY_HEAPS`. Each heap is described by an element of the `memoryHeaps` array as a `VkMemoryHeap` structure. The number of memory types available across all memory heaps is given by `memoryTypeCount` and is less than or equal to `VK_MAX_MEMORY_TYPES`. Each memory type is described by an element of the `memoryTypes` array as a `VkMemoryType` structure.

At least one heap **must** include `VK_MEMORY_HEAP_DEVICE_LOCAL_BIT` in `VkMemoryHeap::flags`. If there are multiple heaps that all have similar performance characteristics, they **may** all include `VK_MEMORY_HEAP_DEVICE_LOCAL_BIT`. In a unified memory architecture (UMA) system there is often only a single memory heap which is considered to be equally “local” to the host and to the device, and such an implementation **must** advertise the heap as device-local.

Each memory type returned by `vkGetPhysicalDeviceMemoryProperties` **must** have its `propertyFlags` set to one of the following values:

- 0
- `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT`
- `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_CACHED_BIT`
- `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_CACHED_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT`
- `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT`
- `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT | VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT`
- `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT | VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_CACHED_BIT`
- `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT | VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_CACHED_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT`
- `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT | VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT`
- `VK_MEMORY_PROPERTY_PROTECTED_BIT`
- `VK_MEMORY_PROPERTY_PROTECTED_BIT | VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT`

There **must** be at least one memory type with both the `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT` and `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` bits set in its `propertyFlags`. There **must** be at least one memory type with the `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT` bit set in its `propertyFlags`.

For each pair of elements **X** and **Y** returned in `memoryTypes`, **X** **must** be placed at a lower index position than **Y** if:

- the set of bit flags returned in the `propertyFlags` member of **X** is a strict subset of the set of bit flags returned in the `propertyFlags` member of **Y**; or
- the `propertyFlags` members of **X** and **Y** are equal, and **X** belongs to a memory heap with greater performance (as determined in an implementation-specific manner)

NOTE

There is no ordering requirement between **X** and **Y** elements for the case their **propertyFlags** members are not in a subset relation. That potentially allows more than one possible way to order the same set of memory types. Notice that the [list of all allowed memory property flag combinations](#) is written in a valid order. But if instead **VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT** was before **VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT**, the list would still be in a valid order.

This ordering requirement enables applications to use a simple search loop to select the desired memory type along the lines of:

```
// Find a memory in 'memoryTypeBitsRequirement' that includes all of
// 'requiredProperties'
int32_t findProperties(const VkPhysicalDeviceMemoryProperties* pMemoryProperties,
                     uint32_t memoryTypeBitsRequirement,
                     VkMemoryPropertyFlags requiredProperties) {
    const uint32_t memoryCount = pMemoryProperties->memoryTypeCount;
    for (uint32_t memoryIndex = 0; memoryIndex < memoryCount; ++memoryIndex) {
        const uint32_t memoryTypeBits = (1 << memoryIndex);
        const bool isRequiredMemoryType = memoryTypeBitsRequirement & memoryTypeBits;

        const VkMemoryPropertyFlags properties =
            pMemoryProperties->memoryTypes[memoryIndex].propertyFlags;
        const bool hasRequiredProperties =
            (properties & requiredProperties) == requiredProperties;

        if (isRequiredMemoryType && hasRequiredProperties)
            return static_cast<int32_t>(memoryIndex);
    }

    // failed to find memory type
    return -1;
}

// Try to find an optimal memory type, or if it does not exist try fallback memory
// type
// 'device' is the VkDevice
// 'image' is the VkImage that requires memory to be bound
// 'memoryProperties' properties as returned by vkGetPhysicalDeviceMemoryProperties
// 'requiredProperties' are the property flags that must be present
// 'optimalProperties' are the property flags that are preferred by the application
VkMemoryRequirements memoryRequirements;
vkGetImageMemoryRequirements(device, image, &memoryRequirements);
int32_t memoryType =
    findProperties(&memoryProperties, memoryRequirements.memoryTypeBits,
optimalProperties);
if (memoryType == -1) // not found; try fallback properties
    memoryType =
        findProperties(&memoryProperties, memoryRequirements.memoryTypeBits,
```



```
requiredProperties);
```

`VK_MAX_MEMORY_TYPES` is the length of an array of `VkMemoryType` structures describing memory types, as returned in `VkPhysicalDeviceMemoryProperties::memoryTypes`.

```
#define VK_MAX_MEMORY_TYPES 32U
```

`VK_MAX_MEMORY_HEAPS` is the length of an array of `VkMemoryHeap` structures describing memory heaps, as returned in `VkPhysicalDeviceMemoryProperties::memoryHeaps`.

```
#define VK_MAX_MEMORY_HEAPS 16U
```

To query memory properties, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceMemoryProperties2(
    VkPhysicalDevice          physicalDevice,
    VkPhysicalDeviceMemoryProperties2* pMemoryProperties);
```

- `physicalDevice` is the handle to the device to query.
- `pMemoryProperties` is a pointer to a `VkPhysicalDeviceMemoryProperties2` structure in which the properties are returned.

`vkGetPhysicalDeviceMemoryProperties2` behaves similarly to `vkGetPhysicalDeviceMemoryProperties`, with the ability to return extended information in a `pNext` chain of output structures.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceMemoryProperties2-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceMemoryProperties2-pMemoryProperties-parameter `pMemoryProperties` **must** be a valid pointer to a `VkPhysicalDeviceMemoryProperties2` structure

The `VkPhysicalDeviceMemoryProperties2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceMemoryProperties2 {
    VkStructureType    sType;
    void*              pNext;
    VkPhysicalDeviceMemoryProperties memoryProperties;
} VkPhysicalDeviceMemoryProperties2;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memoryProperties` is a `VkPhysicalDeviceMemoryProperties` structure which is populated with the same values as in `vkGetPhysicalDeviceMemoryProperties`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMemoryProperties2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PROPERTIES_2`
- VUID-VkPhysicalDeviceMemoryProperties2-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkPhysicalDeviceMemoryBudgetPropertiesEXT`
- VUID-VkPhysicalDeviceMemoryProperties2-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique

The `VkMemoryHeap` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkMemoryHeap {
    VkDeviceSize      size;
    VkMemoryHeapFlags flags;
} VkMemoryHeap;
```

- `size` is the total memory size in bytes in the heap.
- `flags` is a bitmask of `VkMemoryHeapFlagBits` specifying attribute flags for the heap.

Bits which **may** be set in `VkMemoryHeap::flags`, indicating attribute flags for the heap, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkMemoryHeapFlagBits {
    VK_MEMORY_HEAP_DEVICE_LOCAL_BIT = 0x00000001,
    // Provided by VK_VERSION_1_1
    VK_MEMORY_HEAP_MULTI_INSTANCE_BIT = 0x00000002,
    // Provided by VKSC_VERSION_1_0
    VK_MEMORY_HEAP_SEU_SAFE_BIT = 0x00000004,
} VkMemoryHeapFlagBits;
```

- `VK_MEMORY_HEAP_DEVICE_LOCAL_BIT` specifies that the heap corresponds to device-local memory. Device-local memory **may** have different performance characteristics than host-local memory, and **may** support different memory property flags.
- `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` specifies that in a logical device representing more than one physical device, there is a per-physical device instance of the heap memory. By default, an allocation from such a heap will be replicated to each physical device's instance of the heap.

- `VK_MEMORY_HEAP_SEU_SAFE_BIT` specifies that the heap is protected against single event upsets.

NOTE

Many safety critical environments are required to contend with single event upsets (SEUs). It is typical for host memory to include automatic error detection (EDC) or correction (ECC) on platforms where this a concern. `VK_MEMORY_HEAP_SEU_SAFE_BIT` is used to denote device memory heaps that have this protection.

SEU-safe memory **may** have different performance characteristics than SEU-unsafe memory.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkMemoryHeapFlags;
```

`VkMemoryHeapFlags` is a bitmask type for setting a mask of zero or more `VkMemoryHeapFlagBits`.

The `VkMemoryType` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkMemoryType {
    VkMemoryPropertyFlags    propertyFlags;
    uint32_t                 heapIndex;
} VkMemoryType;
```

- `heapIndex` describes which memory heap this memory type corresponds to, and **must** be less than `memoryHeapCount` from the `VkPhysicalDeviceMemoryProperties` structure.
- `propertyFlags` is a bitmask of `VkMemoryPropertyFlagBits` of properties for this memory type.

Bits which **may** be set in `VkMemoryType::propertyFlags`, indicating properties of a memory type, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkMemoryPropertyFlagBits {
    VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT = 0x00000001,
    VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT = 0x00000002,
    VK_MEMORY_PROPERTY_HOST_COHERENT_BIT = 0x00000004,
    VK_MEMORY_PROPERTY_HOST_CACHED_BIT = 0x00000008,
    VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT = 0x00000010,
    // Provided by VK_VERSION_1_1
    VK_MEMORY_PROPERTY_PROTECTED_BIT = 0x00000020,
} VkMemoryPropertyFlagBits;
```

- `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT` bit specifies that memory allocated with this type is the most efficient for device access. This property will be set if and only if the memory type belongs to a heap with the `VK_MEMORY_HEAP_DEVICE_LOCAL_BIT` set.
- `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT` bit specifies that memory allocated with this type **can** be mapped for host access using `vkMapMemory`.

- `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` bit specifies that the host cache management commands `vkFlushMappedMemoryRanges` and `vkInvalidateMappedMemoryRanges` are not needed to manage [availability and visibility](#) on the host.
- `VK_MEMORY_PROPERTY_HOST_CACHED_BIT` bit specifies that memory allocated with this type is cached on the host. Host memory accesses to uncached memory are slower than to cached memory, however uncached memory is always host coherent.
- `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT` bit specifies that the memory type only allows device access to the memory. Memory types **must** not have both `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT` and `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT` set. Additionally, the object's backing memory **may** be provided by the implementation lazily as specified in [Lazily Allocated Memory](#).
- `VK_MEMORY_PROPERTY_PROTECTED_BIT` bit specifies that the memory type only allows device access to the memory, and allows protected queue operations to access the memory. Memory types **must** not have `VK_MEMORY_PROPERTY_PROTECTED_BIT` set and any of `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT` set, or `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` set, or `VK_MEMORY_PROPERTY_HOST_CACHED_BIT` set.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkMemoryPropertyFlags;
```

`VkMemoryPropertyFlags` is a bitmask type for setting a mask of zero or more [VkMemoryPropertyFlagBits](#).

If the `VkPhysicalDeviceMemoryBudgetPropertiesEXT` structure is included in the `pNext` chain of [VkPhysicalDeviceMemoryProperties2](#), it is filled with the current memory budgets and usages.

The `VkPhysicalDeviceMemoryBudgetPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_memory_budget
typedef struct VkPhysicalDeviceMemoryBudgetPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize        heapBudget[VK_MAX_MEMORY_HEAPS];
    VkDeviceSize        heapUsage[VK_MAX_MEMORY_HEAPS];
} VkPhysicalDeviceMemoryBudgetPropertiesEXT;
```

- `sType` is a [VkStructureType](#) value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `heapBudget` is an array of `VK_MAX_MEMORY_HEAPS` [VkDeviceSize](#) values in which memory budgets are returned, with one element for each memory heap. A heap's budget is a rough estimate of how much memory the process **can** allocate from that heap before allocations **may** fail or cause performance degradation. The budget includes any currently allocated device memory.
- `heapUsage` is an array of `VK_MAX_MEMORY_HEAPS` [VkDeviceSize](#) values in which memory usages are returned, with one element for each memory heap. A heap's usage is an estimate of how much memory the process is currently using in that heap.

The values returned in this structure are not invariant. The `heapBudget` and `heapUsage` values **must** be zero for array elements greater than or equal to `VkPhysicalDeviceMemoryProperties::memoryHeapCount`. The `heapBudget` value **must** be non-zero for array elements less than `VkPhysicalDeviceMemoryProperties::memoryHeapCount`. The `heapBudget` value **must** be less than or equal to `VkMemoryHeap::size` for each heap.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMemoryBudgetPropertiesEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_BUDGET_PROPERTIES_EXT`

11.2.2. Device Memory Objects

A Vulkan device operates on data in device memory via memory objects that are represented in the API by a `VkDeviceMemory` handle:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDeviceMemory)
```

Device memory **cannot** be freed [SCID-4]. If `VkPhysicalDeviceVulkanSC10Properties::deviceDestroyFreesMemory` is `VK_TRUE`, the memory is returned to the system when the device is destroyed. Otherwise, it **may** not be returned to the system until the process is terminated.

11.2.3. Device Memory Allocation

To allocate memory objects, call:

```
// Provided by VK_VERSION_1_0
VkResult vkAllocateMemory(
    VkDevice                device,
    const VkMemoryAllocateInfo* pAllocateInfo,
    const VkAllocationCallbacks* pAllocator,
    VkDeviceMemory*          pMemory);
```

- `device` is the logical device that owns the memory.
- `pAllocateInfo` is a pointer to a `VkMemoryAllocateInfo` structure describing parameters of the allocation. A successfully returned allocation **must** use the requested parameters—no substitution is permitted by the implementation.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pMemory` is a pointer to a `VkDeviceMemory` handle in which information about the allocated memory is returned.

Allocations returned by `vkAllocateMemory` are guaranteed to meet any alignment requirement of the implementation. For example, if an implementation requires 128 byte alignment for images and 64 byte alignment for buffers, the device memory returned through this mechanism would be 128-

byte aligned. This ensures that applications **can** correctly suballocate objects of different types (with potentially different alignment requirements) in the same memory object.

When memory is allocated, its contents are undefined with the following constraint:

- The contents of unprotected memory **must** not be a function of the contents of data protected memory objects, even if those memory objects were previously freed.

NOTE	The contents of memory allocated by one application should not be a function of data from protected memory objects of another application, even if those memory objects were previously freed.

The maximum number of valid memory allocations that **can** exist simultaneously within a `VkDevice` **may** be restricted by implementation- or platform-dependent limits. The `maxMemoryAllocationCount` feature describes the number of allocations that **can** exist simultaneously before encountering these internal limits.

NOTE	Many protected memory implementations involve complex hardware and system software support, and often have additional and much lower limits on the number of simultaneous protected memory allocations (from memory types with the <code>VK_MEMORY_PROPERTY_PROTECTED_BIT</code> property) than for non-protected memory allocations. These limits can be system-wide, and depend on a variety of factors outside of the Vulkan implementation, so they cannot be queried in Vulkan. Applications should use as few allocations as possible from such memory types by suballocating aggressively, and be prepared for allocation failure even when there is apparently plenty of capacity remaining in the memory heap. As a guideline, the Vulkan conformance test suite requires that at least 80 minimum-size allocations can exist concurrently when no other uses of protected memory are active in the system.

Some platforms **may** have a limit on the maximum size of a single allocation. For example, certain systems **may** fail to create allocations with a size greater than or equal to 4GB. Such a limit is implementation-dependent, and if such a failure occurs then the error `VK_ERROR_OUT_OF_DEVICE_MEMORY` **must** be returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkAllocateMemory` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkAllocateMemory-pAllocateInfo-01713
`pAllocateInfo->allocationSize` **must** be less than or equal to `VkPhysicalDeviceMemoryProperties::memoryHeaps[memindex].size` where `memindex` = `VkPhysicalDeviceMemoryProperties::memoryTypes[pAllocateInfo->memoryTypeIndex].heapIndex` as returned by `vkGetPhysicalDeviceMemoryProperties` for the `VkPhysicalDevice` that `device` was created from
- VUID-vkAllocateMemory-pAllocateInfo-01714
`pAllocateInfo->memoryTypeIndex` **must** be less than `VkPhysicalDeviceMemoryProperties`

`::memoryTypeCount` as returned by `vkGetPhysicalDeviceMemoryProperties` for the `VkPhysicalDevice` that `device` was created from

- VUID-vkAllocateMemory-maxMemoryAllocationCount-04101
There **must** be less than `VkPhysicalDeviceLimits::maxMemoryAllocationCount` device memory allocations currently allocated on the device
- VUID-vkAllocateMemory-device-05068
The number of device memory objects currently allocated from `device` plus 1 **must** be less than or equal to the total number of device memory objects requested via `VkDeviceObjectReservationCreateInfo::deviceMemoryRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkAllocateMemory-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkAllocateMemory-pAllocateInfo-parameter
`pAllocateInfo` **must** be a valid pointer to a valid `VkMemoryAllocateInfo` structure
- VUID-vkAllocateMemory-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkAllocateMemory-pMemory-parameter
`pMemory` **must** be a valid pointer to a `VkDeviceMemory` handle
- VUID-vkAllocateMemory-device-queuecount
The device **must** have been created with at least 1 queue

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INVALID_EXTERNAL_HANDLE`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkMemoryAllocateInfo` structure is defined as:


```
// Provided by VK_VERSION_1_0
typedef struct VkMemoryAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceSize        allocationSize;
    uint32_t            memoryTypeIndex;
} VkMemoryAllocateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `allocationSize` is the size of the allocation in bytes.
- `memoryTypeIndex` is an index identifying a memory type from the `memoryTypes` array of the `VkPhysicalDeviceMemoryProperties` structure.

The internal data of an allocated device memory object **must** include a reference to implementation-specific resources, referred to as the memory object's *payload*. Applications **can** also import and export that internal data to and from device memory objects to share data between Vulkan instances and other compatible APIs. A `VkMemoryAllocateInfo` structure defines a memory import operation if its `pNext` chain includes one of the following structures:

- `VkImportMemoryFdInfoKHR` with a non-zero `handleType` value
- `VkImportMemoryHostPointerInfoEXT` with a non-zero `handleType` value
- `VkImportMemorySciBufInfoNV` with a non-zero `handleType` value
- `VkImportScreenBufferInfoQNX` with a non-`NULL` `buffer` value

If the parameters define an import operation and the external handle type is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV`, `allocationSize` is ignored. The implementation **must** query the size of this allocation from the `NvSciBufAttrList` associated with the external `NvSciBufObj`.

Whether device memory objects constructed via a memory import operation hold a reference to their payload depends on the properties of the handle type used to perform the import, as defined below for each valid handle type. Importing memory **must** not modify the content of the memory. Implementations **must** ensure that importing memory does not enable the importing Vulkan instance to access any memory or resources in other Vulkan instances other than that corresponding to the memory object imported. Implementations **must** also ensure accessing imported memory which has not been initialized does not allow the importing Vulkan instance to obtain data from the exporting Vulkan instance or vice-versa.

NOTE

How exported and imported memory is isolated is left to the implementation, but applications should be aware that such isolation **may** prevent implementations from placing multiple exportable memory objects in the same physical or virtual page. Hence, applications **should** avoid creating many small external memory objects whenever possible.

Importing memory **must** not increase overall heap usage within a system. However, it **must** affect

the following per-process values:

- `VkPhysicalDeviceLimits::maxMemoryAllocationCount`
- `VkPhysicalDeviceMemoryBudgetPropertiesEXT::heapUsage`

When performing a memory import operation, it is the responsibility of the application to ensure the external handles and their associated payloads meet all valid usage requirements. However, implementations **must** perform sufficient validation of external handles and payloads to ensure that the operation results in a valid memory object which will not cause program termination, device loss, queue stalls, or corruption of other resources when used as allowed according to its allocation parameters. If the external handle provided does not meet these requirements, the implementation **must** fail the memory import operation with the error code `VK_ERROR_INVALID_EXTERNAL_HANDLE`.

Valid Usage

- VUID-VkMemoryAllocateInfo-allocationSize-07897
If the parameters do not define an `import` or `export` operation, `allocationSize` **must** be greater than 0
- VUID-VkMemoryAllocateInfo-None-06657
The parameters **must** not define more than one `import` operation
- VUID-VkMemoryAllocateInfo-allocationSize-07899
If the parameters define an export operation, `allocationSize` **must** be greater than 0
- VUID-VkMemoryAllocateInfo-allocationSize-01742
If the parameters define an import operation, the external handle specified was created by the Vulkan API, and the external handle type is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT`, then the values of `allocationSize` and `memoryTypeIndex` **must** match those specified when the payload being imported was created
- VUID-VkMemoryAllocateInfo-memoryTypeIndex-00648
If the parameters define an import operation and the external handle is a POSIX file descriptor created outside of the Vulkan API, the value of `memoryTypeIndex` **must** be one of those returned by `vkGetMemoryFdPropertiesKHR`
- VUID-VkMemoryAllocateInfo-memoryTypeIndex-01872
If the `protectedMemory` feature is not enabled, the `VkMemoryAllocateInfo::memoryTypeIndex` **must** not indicate a memory type that reports `VK_MEMORY_PROPERTY_PROTECTED_BIT`
- VUID-VkMemoryAllocateInfo-memoryTypeIndex-01744
If the parameters define an import operation and the external handle is a host pointer, the value of `memoryTypeIndex` **must** be one of those returned by `vkGetMemoryHostPointerPropertiesEXT`
- VUID-VkMemoryAllocateInfo-allocationSize-01745
If the parameters define an import operation and the external handle is a host pointer, `allocationSize` **must** be an integer multiple of `VkPhysicalDeviceExternalMemoryHostPropertiesEXT::minImportedHostPointerAlignment`

- VUID-VkMemoryAllocateInfo-screenBufferImport-08941
If the parameters define an import operation and the external handle type is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`, `VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX::screenBufferImport` **must** be enabled
- VUID-VkMemoryAllocateInfo-allocationSize-08942
If the parameters define an import operation and the external handle type is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`, `allocationSize` **must** be the size returned by `vkGetScreenBufferPropertiesQNX` for the QNX Screen buffer
- VUID-VkMemoryAllocateInfo-memoryTypeIndex-08943
If the parameters define an import operation and the external handle type is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`, `memoryTypeIndex` **must** be one of those returned by `vkGetScreenBufferPropertiesQNX` for the QNX Screen buffer
- VUID-VkMemoryAllocateInfo-pNext-08944
If the parameters define an import operation, the external handle is a QNX Screen buffer, and the `pNext` chain includes a `VkMemoryDedicatedAllocateInfo` with `image` that is not `VK_NULL_HANDLE`, the QNX Screen's buffer **must** be a valid QNX Screen buffer
- VUID-VkMemoryAllocateInfo-pNext-08945
If the parameters define an import operation, the external handle is an QNX Screen buffer, and the `pNext` chain includes a `VkMemoryDedicatedAllocateInfo` with `image` that is not `VK_NULL_HANDLE`, the format of `image` **must** be `VK_FORMAT_UNDEFINED` or the format returned by `vkGetScreenBufferPropertiesQNX` in `VkScreenBufferFormatPropertiesQNX::format` for the QNX Screen buffer
- VUID-VkMemoryAllocateInfo-pNext-08946
If the parameters define an import operation, the external handle is a QNX Screen buffer, and the `pNext` chain includes a `VkMemoryDedicatedAllocateInfo` structure with `image` that is not `VK_NULL_HANDLE`, the width, height, and array layer dimensions of `image` and the QNX Screen buffer's `_screen_buffer` **must** be identical
- VUID-VkMemoryAllocateInfo-opaqueCaptureAddress-03329
If `VkMemoryOpaqueCaptureAddressAllocateInfo::opaqueCaptureAddress` is not zero, `VkMemoryAllocateFlagsInfo::flags` **must** include `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`
- VUID-VkMemoryAllocateInfo-flags-03330
If `VkMemoryAllocateFlagsInfo::flags` includes `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`, the `bufferDeviceAddressCaptureReplay` feature **must** be enabled
- VUID-VkMemoryAllocateInfo-flags-03331
If `VkMemoryAllocateFlagsInfo::flags` includes `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT`, the `bufferDeviceAddress` feature **must** be enabled
- VUID-VkMemoryAllocateInfo-pNext-03332
If the `pNext` chain includes a `VkImportMemoryHostPointerInfoEXT` structure, `VkMemoryOpaqueCaptureAddressAllocateInfo::opaqueCaptureAddress` **must** be zero
- VUID-VkMemoryAllocateInfo-opaqueCaptureAddress-03333
If the parameters define an import operation,

`VkMemoryOpaqueCaptureAddressAllocateInfo::opaqueCaptureAddress` **must** be zero

- VUID-VkMemoryAllocateInfo-pNext-05097
If the `pNext` chain includes a `VkExportMemorySciBufInfoNV` structure, the `sciBufExport` feature **must** be enabled
- VUID-VkMemoryAllocateInfo-pNext-05098
If the `pNext` chain includes a `VkImportMemorySciBufInfoNV` structure, the `sciBufImport` feature **must** be enabled
- VUID-VkMemoryAllocateInfo-memoryTypeIndex-05099
If the parameters define an import operation and the external handle is a `NvSciBufObj`, the value of `memoryTypeIndex` **must** be one of those returned by `vkGetPhysicalDeviceExternalMemorySciBufPropertiesNV`

Valid Usage (Implicit)

- VUID-VkMemoryAllocateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO`
- VUID-VkMemoryAllocateInfo-pNext-pNext
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkExportMemoryAllocateInfo`, `VkExportMemorySciBufInfoNV`, `VkImportMemoryFdInfoKHR`, `VkImportMemoryHostPointerInfoEXT`, `VkImportMemorySciBufInfoNV`, `VkImportScreenBufferInfoQNX`, `VkMemoryAllocateFlagsInfo`, `VkMemoryDedicatedAllocateInfo`, or `VkMemoryOpaqueCaptureAddressAllocateInfo`
- VUID-VkMemoryAllocateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique

If the `pNext` chain includes a `VkMemoryDedicatedAllocateInfo` structure, then that structure includes a handle of the sole buffer or image resource that the memory **can** be bound to.

The `VkMemoryDedicatedAllocateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkMemoryDedicatedAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkImage             image;
    VkBuffer            buffer;
} VkMemoryDedicatedAllocateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `image` is `VK_NULL_HANDLE` or a handle of an image which this memory will be bound to.
- `buffer` is `VK_NULL_HANDLE` or a handle of a buffer which this memory will be bound to.

Valid Usage

- VUID-VkMemoryDedicatedAllocateInfo-image-01432
At least one of **image** and **buffer** **must** be [VK_NULL_HANDLE](#)
- VUID-VkMemoryDedicatedAllocateInfo-image-02964
If **image** is not [VK_NULL_HANDLE](#) and the memory is not an imported QNX Screen buffer , [VkMemoryAllocateInfo::allocationSize](#) **must** be greater than or equal to the [VkMemoryRequirements::size](#) of the image
- VUID-VkMemoryDedicatedAllocateInfo-image-01434
If **image** is not [VK_NULL_HANDLE](#), **image** **must** have been created without [VK_IMAGE_CREATE_SPARSE_BINDING_BIT](#) set in [VkImageCreateInfo::flags](#)
- VUID-VkMemoryDedicatedAllocateInfo-buffer-02965
If **buffer** is not [VK_NULL_HANDLE](#) and the memory is not an imported QNX Screen buffer , [VkMemoryAllocateInfo::allocationSize](#) **must** be greater than or equal to the [VkMemoryRequirements::size](#) of the buffer
- VUID-VkMemoryDedicatedAllocateInfo-buffer-01436
If **buffer** is not [VK_NULL_HANDLE](#), **buffer** **must** have been created without [VK_BUFFER_CREATE_SPARSE_BINDING_BIT](#) set in [VkBufferCreateInfo::flags](#)
- VUID-VkMemoryDedicatedAllocateInfo-image-01878
If **image** is not [VK_NULL_HANDLE](#) and [VkMemoryAllocateInfo](#) defines a memory import operation with handle type [VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT](#), the memory being imported **must** also be a dedicated image allocation and **image** **must** be identical to the image associated with the imported memory
- VUID-VkMemoryDedicatedAllocateInfo-buffer-01879
If **buffer** is not [VK_NULL_HANDLE](#) and [VkMemoryAllocateInfo](#) defines a memory import operation with handle type [VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT](#), the memory being imported **must** also be a dedicated buffer allocation and **buffer** **must** be identical to the buffer associated with the imported memory

Valid Usage (Implicit)

- VUID-VkMemoryDedicatedAllocateInfo-sType-sType
sType **must** be [VK_STRUCTURE_TYPE_MEMORY_DEDICATED_ALLOCATE_INFO](#)
- VUID-VkMemoryDedicatedAllocateInfo-image-parameter
If **image** is not [VK_NULL_HANDLE](#), **image** **must** be a valid [VkImage](#) handle
- VUID-VkMemoryDedicatedAllocateInfo-buffer-parameter
If **buffer** is not [VK_NULL_HANDLE](#), **buffer** **must** be a valid [VkBuffer](#) handle
- VUID-VkMemoryDedicatedAllocateInfo-commonparent
Both of **buffer**, and **image** that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same [VkDevice](#)

When allocating memory whose payload **may** be exported to another process or Vulkan instance,

add a [VkExportMemoryAllocateInfo](#) structure to the **pNext** chain of the [VkMemoryAllocateInfo](#) structure, specifying the handle types that **may** be exported.

The [VkExportMemoryAllocateInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExportMemoryAllocateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkExternalMemoryHandleTypeFlags handleTypes;
} VkExportMemoryAllocateInfo;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **handleTypes** is zero or a bitmask of [VkExternalMemoryHandleTypeFlagBits](#) specifying one or more memory handle types the application **can** export from the resulting allocation. The application **can** request multiple handle types for the same allocation.

Valid Usage

- VUID-VkExportMemoryAllocateInfo-handleTypes-09860

The bits in **handleTypes** **must** be supported and compatible, as reported by [VkExternalImageFormatProperties](#), or [VkExternalBufferProperties](#)

Valid Usage (Implicit)

- VUID-VkExportMemoryAllocateInfo-sType-sType
sType **must** be [VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO](#)
- VUID-VkExportMemoryAllocateInfo-handleTypes-parameter
handleTypes **must** be a valid combination of [VkExternalMemoryHandleTypeFlagBits](#) values

11.2.4. File Descriptor External Memory

To import memory from a POSIX file descriptor handle, add a [VkImportMemoryFdInfoKHR](#) structure to the **pNext** chain of the [VkMemoryAllocateInfo](#) structure. The [VkImportMemoryFdInfoKHR](#) structure is defined as:

```
// Provided by VK_KHR_external_memory_fd
typedef struct VkImportMemoryFdInfoKHR {
    VkStructureType           sType;
    const void*               pNext;
    VkExternalMemoryHandleTypeFlagBits handleType;
    int                        fd;
} VkImportMemoryFdInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleType` is a `VkExternalMemoryHandleTypeFlagBits` value specifying the handle type of `fd`.
- `fd` is the external handle to import.

Importing memory from a file descriptor transfers ownership of the file descriptor from the application to the Vulkan implementation. The application **must** not perform any operations on the file descriptor after a successful import. The imported memory object holds a reference to its payload.

Applications **can** import the same payload into multiple instances of Vulkan, into the same instance from which it was exported, and multiple times into a given Vulkan instance. In all cases, each import operation **must** create a distinct `VkDeviceMemory` object.

Valid Usage

- VUID-VkImportMemoryFdInfoKHR-handleType-09862
If `handleType` is not `0`, it **must** be supported for import, as reported by `VkExternalImageFormatProperties` or `VkExternalBufferProperties`
- VUID-VkImportMemoryFdInfoKHR-fd-00668
The memory from which `fd` was exported **must** have been created on the same underlying physical device as `device`
- VUID-VkImportMemoryFdInfoKHR-handleType-00669
If `handleType` is not `0`, it **must** be `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT` or `VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT`
- VUID-VkImportMemoryFdInfoKHR-handleType-00670
If `handleType` is not `0`, `fd` **must** be a valid handle of the type specified by `handleType`
- VUID-VkImportMemoryFdInfoKHR-fd-01746
The memory represented by `fd` **must** have been created from a physical device and driver that is compatible with `device` and `handleType`, as described in [External Memory Handle Types Compatibility](#)
- VUID-VkImportMemoryFdInfoKHR-fd-01520
`fd` **must** obey any requirements listed for `handleType` in [external memory handle types compatibility](#)

Valid Usage (Implicit)

- VUID-VkImportMemoryFdInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_MEMORY_FD_INFO_KHR`
- VUID-VkImportMemoryFdInfoKHR-handleType-parameter
If `handleType` is not `0`, `handleType` **must** be a valid `VkExternalMemoryHandleTypeFlagBits` value

To export a POSIX file descriptor referencing the payload of a Vulkan device memory object, call:

```
// Provided by VK_KHR_external_memory_fd
VkResult vkGetMemoryFdKHR(
    VkDevice device,
    const VkMemoryGetFdInfoKHR* pGetFdInfo,
    int* pFd);
```

- `device` is the logical device that created the device memory being exported.
- `pGetFdInfo` is a pointer to a `VkMemoryGetFdInfoKHR` structure containing parameters of the export operation.
- `pFd` will return a file descriptor referencing the payload of the device memory object.

Each call to `vkGetMemoryFdKHR` **must** create a new file descriptor holding a reference to the memory object's payload and transfer ownership of the file descriptor to the application. To avoid leaking resources, the application **must** release ownership of the file descriptor using the `close` system call when it is no longer needed, or by importing a Vulkan memory object from it. Where supported by the operating system, the implementation **must** set the file descriptor to be closed automatically when an `execve` system call is made.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetMemoryFdKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetMemoryFdKHR-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetMemoryFdKHR-pGetFdInfo-parameter
`pGetFdInfo` **must** be a valid pointer to a valid `VkMemoryGetFdInfoKHR` structure
- VUID-vkGetMemoryFdKHR-pFd-parameter
`pFd` **must** be a valid pointer to an `int` value

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkMemoryGetFdInfoKHR` structure is defined as:

```
// Provided by VK_KHR_external_memory_fd
typedef struct VkMemoryGetFdInfoKHR {
    VkStructureType      sType;
    const void*          pNext;
    VkDeviceMemory       memory;
    VkExternalMemoryHandleTypeFlagBits handleType;
} VkMemoryGetFdInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memory` is the memory object from which the handle will be exported.
- `handleType` is a `VkExternalMemoryHandleTypeFlagBits` value specifying the type of handle requested.

The properties of the file descriptor exported depend on the value of `handleType`. See `VkExternalMemoryHandleTypeFlagBits` for a description of the properties of the defined external memory handle types.

NOTE

The size of the exported file **may** be larger than the size requested by `VkMemoryAllocateInfo::allocationSize`. If `handleType` is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT`, then the application **can** query the file's actual size with `lseek`.

Valid Usage

- VUID-VkMemoryGetFdInfoKHR-handleType-00671
`handleType` **must** have been included in `VkExportMemoryAllocateInfo::handleTypes` when `memory` was created
- VUID-VkMemoryGetFdInfoKHR-handleType-00672
`handleType` **must** be `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT` or

Valid Usage (Implicit)

- VUID-VkMemoryGetFdInfoKHR-sType-sType
sType **must** be `VK_STRUCTURE_TYPE_MEMORY_GET_FD_INFO_KHR`
- VUID-VkMemoryGetFdInfoKHR-pNext-pNext
pNext **must** be `NULL`
- VUID-VkMemoryGetFdInfoKHR-memory-parameter
memory **must** be a valid `VkDeviceMemory` handle
- VUID-VkMemoryGetFdInfoKHR-handleType-parameter
handleType **must** be a valid `VkExternalMemoryHandleTypeFlagBits` value

POSIX file descriptor memory handles compatible with Vulkan **may** also be created by non-Vulkan APIs using methods beyond the scope of this specification. To determine the correct parameters to use when importing such handles, call:

```
// Provided by VK_KHR_external_memory_fd
VkResult vkGetMemoryFdPropertiesKHR(
    VkDevice                device,
    VkExternalMemoryHandleTypeFlagBits handleType,
    int                     fd,
    VkMemoryFdPropertiesKHR* pMemoryFdProperties);
```

- **device** is the logical device that will be importing **fd**.
- **handleType** is a `VkExternalMemoryHandleTypeFlagBits` value specifying the type of the handle **fd**.
- **fd** is the handle which will be imported.
- **pMemoryFdProperties** is a pointer to a `VkMemoryFdPropertiesKHR` structure in which the properties of the handle **fd** are returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetMemoryFdPropertiesKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetMemoryFdPropertiesKHR-fd-00673
fd **must** point to a valid POSIX file descriptor memory handle
- VUID-vkGetMemoryFdPropertiesKHR-handleType-00674
handleType **must** not be `VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT`

Valid Usage (Implicit)

- VUID-vkGetMemoryFdPropertiesKHR-device-parameter **device** must be a valid [VkDevice](#) handle
- VUID-vkGetMemoryFdPropertiesKHR-handleType-parameter **handleType** must be a valid [VkExternalMemoryHandleTypeFlagBits](#) value
- VUID-vkGetMemoryFdPropertiesKHR-pMemoryFdProperties-parameter **pMemoryFdProperties** must be a valid pointer to a [VkMemoryFdPropertiesKHR](#) structure

Return Codes

Success

- [VK_SUCCESS](#)

Failure

- [VK_ERROR_INVALID_EXTERNAL_HANDLE](#)
- [VK_ERROR_OUT_OF_HOST_MEMORY](#)
- [VK_ERROR_UNKNOWN](#)
- [VK_ERROR_VALIDATION_FAILED](#)

The [VkMemoryFdPropertiesKHR](#) structure returned is defined as:

```
// Provided by VK_KHR_external_memory_fd
typedef struct VkMemoryFdPropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           memoryTypeBits;
} VkMemoryFdPropertiesKHR;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is [NULL](#) or a pointer to a structure extending this structure.
- **memoryTypeBits** is a bitmask containing one bit set for every memory type which the specified file descriptor **can** be imported as.

Valid Usage (Implicit)

- VUID-VkMemoryFdPropertiesKHR-sType-sType **sType** must be [VK_STRUCTURE_TYPE_MEMORY_FD_PROPERTIES_KHR](#)
- VUID-VkMemoryFdPropertiesKHR-pNext-pNext **pNext** must be [NULL](#)

11.2.5. Host External Memory

To import memory from a host pointer, add a [VkImportMemoryHostPointerInfoEXT](#) structure to the `pNext` chain of the [VkMemoryAllocateInfo](#) structure. The [VkImportMemoryHostPointerInfoEXT](#) structure is defined as:

```
// Provided by VK_EXT_external_memory_host
typedef struct VkImportMemoryHostPointerInfoEXT {
    VkStructureType          sType;
    const void*              pNext;
    VkExternalMemoryHandleTypeFlagBits handleType;
    void*                    pHostPointer;
} VkImportMemoryHostPointerInfoEXT;
```

- `sType` is a [VkStructureType](#) value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleType` is a [VkExternalMemoryHandleTypeFlagBits](#) value specifying the handle type.
- `pHostPointer` is the host pointer to import from.

Importing memory from a host pointer shares ownership of the memory between the host and the Vulkan implementation. The application **can** continue to access the memory through the host pointer but it is the application's responsibility to synchronize device and non-device access to the payload as defined in [Host Access to Device Memory Objects](#).

Applications **can** import the same payload into multiple instances of Vulkan and multiple times into a given Vulkan instance. However, implementations **may** fail to import the same payload multiple times into a given physical device due to platform constraints.

Importing memory from a particular host pointer **may** not be possible due to additional platform-specific restrictions beyond the scope of this specification in which case the implementation **must** fail the memory import operation with the error code `VK_ERROR_INVALID_EXTERNAL_HANDLE_KHR`.

Whether device memory objects imported from a host pointer hold a reference to their payload is undefined. As such, the application **must** ensure that the imported memory range remains valid and accessible for the lifetime of the imported memory object.

Implementations **may** support importing host pointers for memory types which are not host-visible. In this case, after a successful call to [vkAllocateMemory](#), the memory range imported from `pHostPointer` **must** not be accessed by the application until the [VkDeviceMemory](#) has been destroyed. Memory contents for the host memory becomes undefined on import, and is left undefined after the [VkDeviceMemory](#) has been destroyed. Applications **must** also not access host memory which is mapped to the same physical memory as `pHostPointer`, but mapped to a different host pointer while the [VkDeviceMemory](#) handle is valid. Implementations running on general-purpose operating systems **should** not support importing host pointers for memory types which are not host-visible.

NOTE

Using host pointers to back non-host visible allocations is a platform-specific use

case, and applications should not attempt to do this unless instructed by the platform.

Valid Usage

- VUID-VkImportMemoryHostPointerInfoEXT-handleType-01747
If `handleType` is not `0`, it **must** be supported for import, as reported in [VkExternalMemoryProperties](#)
- VUID-VkImportMemoryHostPointerInfoEXT-handleType-01748
If `handleType` is not `0`, it **must** be `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT` or `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT`
- VUID-VkImportMemoryHostPointerInfoEXT-pHostPointer-01749
`pHostPointer` **must** be a pointer aligned to an integer multiple of `VkPhysicalDeviceExternalMemoryHostPropertiesEXT::minImportedHostPointerAlignment`
- VUID-VkImportMemoryHostPointerInfoEXT-handleType-01750
If `handleType` is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT`, `pHostPointer` **must** be a pointer to `allocationSize` number of bytes of host memory, where `allocationSize` is the member of the `VkMemoryAllocateInfo` structure this structure is chained to
- VUID-VkImportMemoryHostPointerInfoEXT-handleType-01751
If `handleType` is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT`, `pHostPointer` **must** be a pointer to `allocationSize` number of bytes of host mapped foreign memory, where `allocationSize` is the member of the `VkMemoryAllocateInfo` structure this structure is chained to

Valid Usage (Implicit)

- VUID-VkImportMemoryHostPointerInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_MEMORY_HOST_POINTER_INFO_EXT`
- VUID-VkImportMemoryHostPointerInfoEXT-handleType-parameter
`handleType` **must** be a valid [VkExternalMemoryHandleTypeFlagBits](#) value
- VUID-VkImportMemoryHostPointerInfoEXT-pHostPointer-parameter
`pHostPointer` **must** be a pointer value

To determine the correct parameters to use when importing host pointers, call:

```
// Provided by VK_EXT_external_memory_host
VkResult vkGetMemoryHostPointerPropertiesEXT(
    VkDevice                                device,
    VkExternalMemoryHandleTypeFlagBits      handleType,
    const void*                             pHostPointer,
    VkMemoryHostPointerPropertiesEXT*        pMemoryHostPointerProperties);
```

- `device` is the logical device that will be importing `pHostPointer`.
- `handleType` is a `VkExternalMemoryHandleTypeFlagBits` value specifying the type of the handle `pHostPointer`.
- `pHostPointer` is the host pointer to import from.
- `pMemoryHostPointerProperties` is a pointer to a `VkMemoryHostPointerPropertiesEXT` structure in which the host pointer properties are returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetMemoryHostPointerPropertiesEXT` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetMemoryHostPointerPropertiesEXT-handleType-01752
`handleType` **must** be `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT` or `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT`
- VUID-vkGetMemoryHostPointerPropertiesEXT-pHostPointer-01753
`pHostPointer` **must** be a pointer aligned to an integer multiple of `VkPhysicalDeviceExternalMemoryHostPropertiesEXT::minImportedHostPointerAlignment`
- VUID-vkGetMemoryHostPointerPropertiesEXT-handleType-01754
If `handleType` is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT`, `pHostPointer` **must** be a pointer to host memory
- VUID-vkGetMemoryHostPointerPropertiesEXT-handleType-01755
If `handleType` is `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT`, `pHostPointer` **must** be a pointer to host mapped foreign memory

Valid Usage (Implicit)

- VUID-vkGetMemoryHostPointerPropertiesEXT-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetMemoryHostPointerPropertiesEXT-handleType-parameter
`handleType` **must** be a valid `VkExternalMemoryHandleTypeFlagBits` value
- VUID-vkGetMemoryHostPointerPropertiesEXT-pHostPointer-parameter
`pHostPointer` **must** be a pointer value
- VUID-vkGetMemoryHostPointerPropertiesEXT-pMemoryHostPointerProperties-parameter
`pMemoryHostPointerProperties` **must** be a valid pointer to a `VkMemoryHostPointerPropertiesEXT` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INVALID_EXTERNAL_HANDLE`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkMemoryHostPointerPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_external_memory_host
typedef struct VkMemoryHostPointerPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           memoryTypeBits;
} VkMemoryHostPointerPropertiesEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memoryTypeBits` is a bitmask containing one bit set for every memory type which the specified host pointer **can** be imported as.

The value returned by `memoryTypeBits` **should** only include bits that identify memory types which are host visible. Implementations **may** include bits that identify memory types which are not host visible. Behavior for imported pointers of such types is defined by `VkImportMemoryHostPointerInfoEXT`.

Valid Usage (Implicit)

- VUID-VkMemoryHostPointerPropertiesEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_HOST_POINTER_PROPERTIES_EXT`
- VUID-VkMemoryHostPointerPropertiesEXT-pNext-pNext
`pNext` **must** be `NULL`

11.2.6. NvSciBuf External Memory

To export a `NvSciBufObj` from memory, add a `VkExportMemorySciBufInfoNV` structure to the `pNext` chain of the `VkMemoryAllocateInfo` structure. The `VkExportMemorySciBufInfoNV` structure is defined as:

```
// Provided by VK_NV_external_memory_sci_buf
typedef struct VkExportMemorySciBufInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    NvSciBufAttrList    pAttributes;
} VkExportMemorySciBufInfoNV;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pAttributes` is an opaque `NvSciBufAttrList` describing the attributes of the `NvSciBuf` object that will be exported.

If `VkExportMemoryAllocateInfo` is not present in the same `pNext` chain, this structure is ignored.

If the `pNext` chain of `VkMemoryAllocateInfo` includes a `VkExportMemoryAllocateInfo` structure with a `handleType` mask containing the `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV` bit, but either `VkExportMemorySciBufInfoNV` is not included in the `pNext` chain, or it is included but `pAttributes` is `NULL`, `vkAllocateMemory` will return `VK_ERROR_INITIALIZATION_FAILED`.

The `pAttributes` parameter **must** be a reconciled `NvSciBufAttrList`. `NvSciBufAttrList` consists of both public and private attributes. It is the application's responsibility to set the public attributes. To set the private attributes, the application **must** use the `vkGetPhysicalDeviceSciBufAttributesNV` command. The `NvSciBufAttrList` is then reconciled using the `NvSciBuf` APIs.

Valid Usage

- VUID-VkExportMemorySciBufInfoNV-pAttributes-05100
`pAttributes` **must** be a reconciled `NvSciBufAttrList`

Valid Usage (Implicit)

- VUID-VkExportMemorySciBufInfoNV-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_EXPORT_MEMORY_SCI_BUF_INFO_NV`

To fill the private attributes of an unreconciled `NvSciBufAttrList`, call:

```
// Provided by VK_NV_external_memory_sci_buf
VkResult vkGetPhysicalDeviceSciBufAttributesNV(
    VkPhysicalDevice    physicalDevice,
    NvSciBufAttrList     pAttributes);
```

- `physicalDevice` is the handle to the physical device that will be used to determine the attributes.
- `pAttributes` is an opaque `NvSciBufAttrList` in which the implementation will set the requested attributes.

On success, `pAttributes` will contain an unreconciled `NvSciBufAttrList` whose private attributes are filled in by the implementation. If the private attributes of `physicalDevice` could not be obtained, `VK_ERROR_INITIALIZATION_FAILED` is returned.

Valid Usage

- VUID-vkGetPhysicalDeviceSciBufAttributesNV-pAttributes-05101
`pAttributes` **must** be a valid `NvSciBufAttrList` and **must** not be `NULL`

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSciBufAttributesNV-physicalDevice-parameter
`physicalDevice` **must** be a valid `VkPhysicalDevice` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To import memory from a `NvSciBufObj`, add a `VkImportMemorySciBufInfoNV` structure to the `pNext` chain of the `VkMemoryAllocateInfo` structure.

The `VkImportMemorySciBufInfoNV` structure is defined as:

```
// Provided by VK_NV_external_memory_sci_buf
typedef struct VkImportMemorySciBufInfoNV {
    VkStructureType           sType;
    const void*               pNext;
    VkExternalMemoryHandleTypeFlagBits handleType;
    NvSciBufObj               handle;
} VkImportMemorySciBufInfoNV;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleType` specifies the type of handle or name.
- `handle` is the external handle to import.

Importing memory from a `NvSciBufObj` does not transfer ownership of the `NvSciBufObj` from the application to the Vulkan implementation. Vulkan will increment the reference count of the underlying memory of the imported `NvSciBufObj`. The application **must** release its ownership using `NvSciBuf APIs` when that ownership is no longer needed.

Applications **can** import the same payload into multiple instances of Vulkan, into the same instance from which it was exported, and multiple times into a given Vulkan instance. In all cases, each import operation **must** create a distinct `VkDeviceMemory` object.

After successfully importing the `NvSciBufObj` to `VkDeviceMemory`, the application **can** use it as a normal `VkDeviceMemory` object. It is the application's responsibility to synchronize the different `NvSciBufObj` accesses.

Valid Usage

- VUID-VkImportMemorySciBufInfoNV-handleType-05102
`handleType` **must** be `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV`

Valid Usage (Implicit)

- VUID-VkImportMemorySciBufInfoNV-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_MEMORY_SCI_BUF_INFO_NV`
- VUID-VkImportMemorySciBufInfoNV-handleType-parameter
`handleType` **must** be a valid `VkExternalMemoryHandleTypeFlagBits` value

To export a `NvSciBufObj` representing the payload of a Vulkan device memory object, call:

```
// Provided by VK_NV_external_memory_sci_buf
VkResult vkGetMemorySciBufNV(
    VkDevice          device,
    const VkMemoryGetSciBufInfoNV* pGetSciBufInfo,
    NvSciBufObj*      pHandle);
```

- `device` is the logical device that created the device memory being exported.
- `pGetSciBufInfo` is a pointer to a `VkMemoryGetSciBufInfoNV` structure containing parameters of the export operation.
- `pHandle` will return the `NvSciBufObj` representing the payload of the device memory object.

A call to `vkGetMemorySciBufNV` will not transfer the ownership of the `NvSciBufObj` handle to the application. The application will hold a reference to the `NvSciBufObj`, but it does not add a reference count to the `NvSciBufObj`, so the application **must** not release it.

Valid Usage (Implicit)

- VUID-vkGetMemorySciBufNV-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkGetMemorySciBufNV-pGetSciBufInfo-parameter
pGetSciBufInfo **must** be a valid pointer to a valid [VkMemoryGetSciBufInfoNV](#) structure
- VUID-vkGetMemorySciBufNV-pHandle-parameter
pHandle **must** be a valid pointer to a [NvSciBufObj](#) value

Return Codes

Success

- [VK_SUCCESS](#)

Failure

- [VK_ERROR_INITIALIZATION_FAILED](#)
- [VK_ERROR_UNKNOWN](#)
- [VK_ERROR_VALIDATION_FAILED](#)

The [VkMemoryGetSciBufInfoNV](#) structure is defined as:

```
// Provided by VK_NV_external_memory_sci_buf
typedef struct VkMemoryGetSciBufInfoNV {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceMemory      memory;
    VkExternalMemoryHandleTypeFlagBits handleType;
} VkMemoryGetSciBufInfoNV;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is [NULL](#) or a pointer to a structure extending this structure.
- **memory** is the memory object from which the handle will be exported.
- **handleType** is the type of handle requested.

Valid Usage

- VUID-VkMemoryGetSciBufInfoNV-handleType-05103
handleType **must** be [VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV](#)

Valid Usage (Implicit)

- VUID-VkMemoryGetSciBufInfoNV-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_GET_SCI_BUF_INFO_NV`
- VUID-VkMemoryGetSciBufInfoNV-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkMemoryGetSciBufInfoNV-memory-parameter
`memory` **must** be a valid `VkDeviceMemory` handle
- VUID-VkMemoryGetSciBufInfoNV-handleType-parameter
`handleType` **must** be a valid `VkExternalMemoryHandleTypeFlagBits` value

A `NvSciBufObj` handle compatible with Vulkan **can** also be created by non-Vulkan APIs using methods beyond the scope of this specification. To determine the correct parameters to use when importing such handles, call:

```
// Provided by VK_NV_external_memory_sci_buf
VkResult vkGetPhysicalDeviceExternalMemorySciBufPropertiesNV(
    VkPhysicalDevice          physicalDevice,
    VkExternalMemoryHandleTypeFlagBits handleType,
    NvSciBufObj               handle,
    VkMemorySciBufPropertiesNV* pMemorySciBufProperties);
```

- `physicalDevice` is the handle to the physical device whose properties will be queried.
- `handleType` is the type of the handle `handle`.
- `handle` is the `NvSciBufObj` handle which will be imported.
- `pMemorySciBufProperties` is a pointer to a `VkMemorySciBufPropertiesNV` structure.

This command will return properties of `handle`, it contains the memory type bitmask that **can** be used to determine the `VkMemoryAllocateInfo::memoryTypeIndex` when calling `vkAllocateMemory`.

Valid Usage

- VUID-vkGetPhysicalDeviceExternalMemorySciBufPropertiesNV-handleType-05104
`handleType` **must** be `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV`
- VUID-vkGetPhysicalDeviceExternalMemorySciBufPropertiesNV-sciBufImport-05105
`VkPhysicalDeviceExternalMemorySciBufFeaturesNV::sciBufImport` **must** be enabled

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceExternalMemorySciBufPropertiesNV-physicalDevice-parameter
`physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceExternalMemorySciBufPropertiesNV-handleType-parameter

`handleType` **must** be a valid `VkExternalMemoryHandleTypeFlagBits` value

- VUID-vkGetPhysicalDeviceExternalMemorySciBufPropertiesNV-pMemorySciBufProperties-parameter

`pMemorySciBufProperties` **must** be a valid pointer to a `VkMemorySciBufPropertiesNV` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_INVALID_EXTERNAL_HANDLE`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkMemorySciBufPropertiesNV` structure is defined as:

```
// Provided by VK_NV_external_memory_sci_buf
typedef struct VkMemorySciBufPropertiesNV {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           memoryTypeBits;
} VkMemorySciBufPropertiesNV;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memoryTypeBits` is a bitmask containing one bit set for every memory type for which the specified `NvSciBufObj` handle **can** be imported.

Valid Usage (Implicit)

- VUID-VkMemorySciBufPropertiesNV-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_SCI_BUF_PROPERTIES_NV`
- VUID-VkMemorySciBufPropertiesNV-pNext-pNext
`pNext` **must** be `NULL`

11.2.7. QNX Screen Buffer External Memory

To import memory created outside of the current Vulkan instance from a QNX Screen buffer, add a `VkImportScreenBufferInfoQNX` structure to the `pNext` chain of the `VkMemoryAllocateInfo` structure.

The `VkImportScreenBufferInfoQNX` structure is defined as:

```
// Provided by VK_QNX_external_memory_screen_buffer
typedef struct VkImportScreenBufferInfoQNX {
    VkStructureType      sType;
    const void*          pNext;
    struct _screen_buffer* buffer;
} VkImportScreenBufferInfoQNX;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `buffer` is a pointer to a `struct _screen_buffer`, the QNX Screen buffer to import

The implementation **may** not acquire a reference to the imported Screen buffer. Therefore, the application **must** ensure that the object referred to by `buffer` stays valid as long as the device memory to which it is imported is being used.

Valid Usage

- VUID-VkImportScreenBufferInfoQNX-buffer-08966
If `buffer` is not `NULL`, QNX Screen Buffers **must** be supported for import, as reported by `VkExternalImageFormatProperties` or `VkExternalBufferProperties`
- VUID-VkImportScreenBufferInfoQNX-buffer-08967
`buffer` is not `NULL`, it **must** be a pointer to `valid QNX Screen buffer`

Valid Usage (Implicit)

- VUID-VkImportScreenBufferInfoQNX-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMPORT_SCREEN_BUFFER_INFO_QNX`

To determine the memory parameters to use when importing a QNX Screen buffer, call:

```
// Provided by VK_QNX_external_memory_screen_buffer
VkResult vkGetScreenBufferPropertiesQNX(
    VkDevice          device,
    const struct _screen_buffer* buffer,
    VkScreenBufferPropertiesQNX* pProperties);
```

- `device` is the logical device that will be importing `buffer`.
- `buffer` is the QNX Screen buffer which will be imported.
- `pProperties` is a pointer to a `VkScreenBufferPropertiesQNX` structure in which the properties of `buffer` are returned.

Valid Usage

- VUID-vkGetScreenBufferPropertiesQNX-buffer-08968
buffer must be a [valid QNX Screen buffer](#)

Valid Usage (Implicit)

- VUID-vkGetScreenBufferPropertiesQNX-device-parameter
device must be a valid [VkDevice](#) handle
- VUID-vkGetScreenBufferPropertiesQNX-buffer-parameter
buffer must be a valid pointer to a valid [_screen_buffer](#) value
- VUID-vkGetScreenBufferPropertiesQNX-pProperties-parameter
pProperties must be a valid pointer to a [VkScreenBufferPropertiesQNX](#) structure

Return Codes

Success

- [VK_SUCCESS](#)

Failure

- [VK_ERROR_OUT_OF_HOST_MEMORY](#)
- [VK_ERROR_UNKNOWN](#)
- [VK_ERROR_VALIDATION_FAILED](#)

The [VkScreenBufferPropertiesQNX](#) structure returned is defined as:

```
// Provided by VK_QNX_external_memory_screen_buffer
typedef struct VkScreenBufferPropertiesQNX {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize        allocationSize;
    uint32_t            memoryTypeBits;
} VkScreenBufferPropertiesQNX;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is [NULL](#) or a pointer to a structure extending this structure.
- **allocationSize** is the size of the external memory.
- **memoryTypeBits** is a bitmask containing one bit set for every memory type which the specified Screen buffer **can** be imported as.

Valid Usage (Implicit)

- VUID-VkScreenBufferPropertiesQNX-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SCREEN_BUFFER_PROPERTIES_QNX`
- VUID-VkScreenBufferPropertiesQNX-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkScreenBufferFormatPropertiesQNX`
- VUID-VkScreenBufferPropertiesQNX-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique

To obtain format properties of a QNX Screen buffer, include a `VkScreenBufferFormatPropertiesQNX` structure in the `pNext` chain of the `VkScreenBufferPropertiesQNX` structure passed to `vkGetScreenBufferPropertiesQNX`. This structure is defined as:

```
// Provided by VK_QNX_external_memory_screen_buffer
typedef struct VkScreenBufferFormatPropertiesQNX {
    VkStructureType    sType;
    void*              pNext;
    VkFormat            format;
    uint64_t           externalFormat;
    uint64_t           screenUsage;
    VkFormatFeatureFlags formatFeatures;
    VkComponentMapping  samplerYcbcrConversionComponents;
    VkSamplerYcbcrModelConversion suggestedYcbcrModel;
    VkSamplerYcbcrRange suggestedYcbcrRange;
    VkChromaLocation    suggestedXChromaOffset;
    VkChromaLocation    suggestedYChromaOffset;
} VkScreenBufferFormatPropertiesQNX;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `format` is the Vulkan format corresponding to the Screen buffer's format or `VK_FORMAT_UNDEFINED` if there is not an equivalent Vulkan format.
- `externalFormat` is an implementation-defined external format identifier for use with `VkExternalFormatQNX`. It **must** not be zero.
- `screenUsage` is an implementation-defined external usage identifier for the QNX Screen buffer.
- `formatFeatures` describes the capabilities of this external format when used with an image bound to memory imported from `buffer`.
- `samplerYcbcrConversionComponents` is the component swizzle that **should** be used in `VkSamplerYcbcrConversionCreateInfo`.
- `suggestedYcbcrModel` is a suggested color model to use in the `VkSamplerYcbcrConversionCreateInfo`.
- `suggestedYcbcrRange` is a suggested numerical value range to use in

[VkSamplerYcbcrConversionCreateInfo](#).

- `suggestedXChromaOffset` is a suggested X chroma offset to use in [VkSamplerYcbcrConversionCreateInfo](#).
- `suggestedYChromaOffset` is a suggested Y chroma offset to use in [VkSamplerYcbcrConversionCreateInfo](#).

If the QNX Screen buffer has one of the formats listed in the [QNX Screen Format Equivalence table](#), then `format` **must** have the equivalent Vulkan format listed in the table. Otherwise, `format` **may** be `VK_FORMAT_UNDEFINED`, indicating the QNX Screen buffer **can** only be used with an external format. The `formatFeatures` member **must** include `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT` and **should** include `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT`.

Valid Usage (Implicit)

- `VUID-VkScreenBufferFormatPropertiesQNX-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_SCREEN_BUFFER_FORMAT_PROPERTIES_QNX`

11.2.8. Device Group Memory Allocations

If the `pNext` chain of [VkMemoryAllocateInfo](#) includes a [VkMemoryAllocateFlagsInfo](#) structure, then that structure includes flags and a device mask controlling how many instances of the memory will be allocated.

The [VkMemoryAllocateFlagsInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkMemoryAllocateFlagsInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkMemoryAllocateFlags flags;
    uint32_t           deviceMask;
} VkMemoryAllocateFlagsInfo;
```

- `sType` is a [VkStructureType](#) value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of [VkMemoryAllocateFlagBits](#) controlling the allocation.
- `deviceMask` is a mask of physical devices in the logical device, indicating that memory **must** be allocated on each device in the mask, if `VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT` is set in `flags`.

If `VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT` is not set, the number of instances allocated depends on whether `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` is set in the memory heap. If `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` is set, then memory is allocated for every physical device in the logical device (as if `deviceMask` has bits set for all device indices). If `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` is not set, then a single instance of memory is allocated (as if

`deviceMask` is set to one).

On some implementations, allocations from a multi-instance heap **may** consume memory on all physical devices even if the `deviceMask` excludes some devices. If `VkPhysicalDeviceGroupProperties::subsetAllocation` is `VK_TRUE`, then memory is only consumed for the devices in the device mask.

NOTE

In practice, most allocations on a multi-instance heap will be allocated across all physical devices. Unicast allocation support is an optional optimization for a minority of allocations.

Valid Usage

- VUID-VkMemoryAllocateFlagsInfo-deviceMask-00675
If `VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT` is set, `deviceMask` **must** be a valid device mask
- VUID-VkMemoryAllocateFlagsInfo-deviceMask-00676
If `VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT` is set, `deviceMask` **must** not be zero

Valid Usage (Implicit)

- VUID-VkMemoryAllocateFlagsInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_FLAGS_INFO`
- VUID-VkMemoryAllocateFlagsInfo-flags-parameter
`flags` **must** be a valid combination of `VkMemoryAllocateFlagBits` values

Bits which **can** be set in `VkMemoryAllocateFlagsInfo::flags`, controlling device memory allocation, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkMemoryAllocateFlagBits {
    VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT = 0x00000001,
    // Provided by VK_VERSION_1_2
    VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT = 0x00000002,
    // Provided by VK_VERSION_1_2
    VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT = 0x00000004,
} VkMemoryAllocateFlagBits;
```

- `VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT` specifies that memory will be allocated for the devices in `VkMemoryAllocateFlagsInfo::deviceMask`.
- `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT` specifies that the memory **can** be attached to a buffer object created with the `VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT` bit set in `usage`, and that the memory handle **can** be used to retrieve an opaque address via `vkGetDeviceMemoryOpaqueCaptureAddress`.
- `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT` specifies that the memory's address **can**

be saved and reused on a subsequent run (e.g. for trace capture and replay), see [VkBufferOpaqueCaptureAddressCreateInfo](#) for more detail.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkMemoryAllocateFlags;
```

[VkMemoryAllocateFlags](#) is a bitmask type for setting a mask of zero or more [VkMemoryAllocateFlagBits](#).

11.2.9. Opaque Capture Address Allocation

To request a specific device address for a memory allocation, add a [VkMemoryOpaqueCaptureAddressAllocateInfo](#) structure to the [pNext](#) chain of the [VkMemoryAllocateInfo](#) structure. The [VkMemoryOpaqueCaptureAddressAllocateInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkMemoryOpaqueCaptureAddressAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint64_t           opaqueCaptureAddress;
} VkMemoryOpaqueCaptureAddressAllocateInfo;
```

- [sType](#) is a [VkStructureType](#) value identifying this structure.
- [pNext](#) is `NULL` or a pointer to a structure extending this structure.
- [opaqueCaptureAddress](#) is the opaque capture address requested for the memory allocation.

If [opaqueCaptureAddress](#) is zero, no specific address is requested.

If [opaqueCaptureAddress](#) is not zero, it **should** be an address retrieved from [vkGetDeviceMemoryOpaqueCaptureAddress](#) on an identically created memory allocation on the same implementation.

NOTE

In most cases, it is expected that a non-zero [opaqueAddress](#) is an address retrieved from [vkGetDeviceMemoryOpaqueCaptureAddress](#) on an identically created memory allocation. If this is not the case, it is likely that [VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS](#) errors will occur.

This is, however, not a strict requirement because trace capture/replay tools may need to adjust memory allocation parameters for imported memory.

If this structure is not present, it is as if [opaqueCaptureAddress](#) is zero.

Valid Usage (Implicit)

- VUID-VkMemoryOpaqueCaptureAddressAllocateInfo-sType-sType

sType **must** be VK_STRUCTURE_TYPE_MEMORY_OPAQUE_CAPTURE_ADDRESS_ALLOCATE_INFO

11.2.10. Host Access to Device Memory Objects

Memory objects created with `vkAllocateMemory` are not directly host accessible.

Memory objects created with the memory property `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT` are considered *mappable*. Memory objects **must** be mappable in order to be successfully mapped on the host.

To retrieve a host virtual address pointer to a region of a mappable memory object, call:

```
// Provided by VK_VERSION_1_0
VkResult vkMapMemory(
    VkDevice          device,
    VkDeviceMemory    memory,
    VkDeviceSize      offset,
    VkDeviceSize      size,
    VkMemoryMapFlags  flags,
    void**            ppData);
```

- `device` is the logical device that owns the memory.
- `memory` is the `VkDeviceMemory` object to be mapped.
- `offset` is a zero-based byte offset from the beginning of the memory object.
- `size` is the size of the memory range to map, or `VK_WHOLE_SIZE` to map from `offset` to the end of the allocation.
- `flags` is reserved for future use.
- `ppData` is a pointer to a `void*` variable in which a host-accessible pointer to the beginning of the mapped range is returned. The value of the returned pointer minus `offset` **must** be aligned to `VkPhysicalDeviceLimits::minMemoryMapAlignment`.

After a successful call to `vkMapMemory` the memory object `memory` is considered to be currently *host mapped*.

NOTE

It is an application error to call `vkMapMemory` on a memory object that is already *host mapped*.

NOTE

`vkMapMemory` will fail if the implementation is unable to allocate an appropriately sized contiguous virtual address range, e.g. due to virtual address space fragmentation or platform limits. In such cases, `vkMapMemory` **must** return `VK_ERROR_MEMORY_MAP_FAILED`. The application **can** improve the likelihood of success by reducing the size of the mapped range and/or removing unneeded mappings using `vkUnmapMemory`.

`vkMapMemory` does not check whether the device memory is currently in use before returning the

host-accessible pointer. The application **must** guarantee that any previously submitted command that writes to this range has completed before the host reads from or writes to that range, and that any previously submitted command that reads from that range has completed before the host writes to that region (see [here](#) for details on fulfilling such a guarantee). If the device memory was allocated without the `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` set, these guarantees **must** be made for an extended range: the application **must** round down the start of the range to the nearest multiple of `VkPhysicalDeviceLimits::nonCoherentAtomSize`, and round the end of the range up to the nearest multiple of `VkPhysicalDeviceLimits::nonCoherentAtomSize`.

While a range of device memory is host mapped, the application is responsible for synchronizing both device and host access to that memory range.

NOTE

It is important for the application developer to become meticulously familiar with all of the mechanisms described in the chapter on [Synchronization and Cache Control](#) as they are crucial to maintaining memory access ordering.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkMapMemory` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkMapMemory-memory-00678
`memory` **must** not be currently host mapped
- VUID-vkMapMemory-offset-00679
`offset` **must** be less than the size of `memory`
- VUID-vkMapMemory-size-00680
If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be greater than 0
- VUID-vkMapMemory-size-00681
If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be less than or equal to the size of the `memory` minus `offset`
- VUID-vkMapMemory-memory-00682
`memory` **must** have been created with a memory type that reports `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT`

Valid Usage (Implicit)

- VUID-vkMapMemory-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkMapMemory-memory-parameter
`memory` **must** be a valid `VkDeviceMemory` handle
- VUID-vkMapMemory-flags-zero bitmask
`flags` **must** be 0
- VUID-vkMapMemory-ppData-parameter
`ppData` **must** be a valid pointer to a pointer value

- VUID-vkMapMemory-memory-parent
`memory` **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `memory` **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_MEMORY_MAP_FAILED`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkMemoryMapFlags;
```

`VkMemoryMapFlags` is a bitmask type for setting a mask of zero or more `VkMemoryMapFlagBits`.

Two commands are provided to enable applications to work with non-coherent memory allocations: `vkFlushMappedMemoryRanges` and `vkInvalidateMappedMemoryRanges`.

NOTE

If the memory object was created with the `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` set, `vkFlushMappedMemoryRanges` and `vkInvalidateMappedMemoryRanges` are unnecessary and **may** have a performance cost. However, [availability and visibility operations](#) still need to be managed on the device. See the description of [host access types](#) for more information.

NOTE

While memory objects imported from a handle type of `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT` or `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT` are inherently mapped to host address space, they are not considered to be host mapped device memory unless they are explicitly host mapped using `vkMapMemory`. That means flushing or invalidating host caches with respect to host accesses performed on such memory through the original host pointer specified at import time is the responsibility of the application and **must** be performed with appropriate synchronization primitives provided by the platform which are outside the scope of Vulkan. `vkFlushMappedMemoryRanges` and `vkInvalidateMappedMemoryRanges`, however,

can still be used on such memory objects to synchronize host accesses performed through the host pointer of the host mapped device memory range returned by `vkMapMemory`.

After a successful call to `vkMapMemory` the memory object `memory` is considered to be currently *host mapped*.

To flush ranges of non-coherent memory from the host caches, call:

```
// Provided by VK_VERSION_1_0
VkResult vkFlushMappedMemoryRanges(
    VkDevice          device,
    uint32_t          memoryRangeCount,
    const VkMappedMemoryRange* pMemoryRanges);
```

- `device` is the logical device that owns the memory ranges.
- `memoryRangeCount` is the length of the `pMemoryRanges` array.
- `pMemoryRanges` is a pointer to an array of `VkMappedMemoryRange` structures describing the memory ranges to flush.

`vkFlushMappedMemoryRanges` guarantees that host writes to the memory ranges described by `pMemoryRanges` are made available to the host memory domain, such that they **can** be made available to the device memory domain via `memory domain operations` using the `VK_ACCESS_HOST_WRITE_BIT` access type.

The first `synchronization scope` includes all host operations that happened-before it, as defined by the host memory model.

NOTE

Some systems allow writes that do not directly integrate with the host memory model; these have to be synchronized by the application manually. One example of this is non-temporal store instructions on x86; to ensure these happen-before submission, applications should call `_mm_sfence()`.

The second `synchronization scope` is empty.

The first `access scope` includes host writes to the specified memory ranges.

NOTE

When a host write to a memory location is made available in this way, each whole aligned set of `nonCoherentAtomSize` bytes that the memory location exists in will also be made available as if they were written by the host. For example, with a `nonCoherentAtomSize` of 128, if an application writes to the first byte of a memory object via a host mapping, the first 128 bytes of the memory object will be made available by this command. While the value of the following 127 bytes will be unchanged, this does count as an access for the purpose of synchronization, so care must be taken to avoid data races.

The second `access scope` is empty.

Unmapping non-coherent memory does not implicitly flush the host mapped memory, and host writes that have not been flushed **may** not ever be visible to the device. However, implementations **must** ensure that writes that have not been flushed do not become visible to any other memory.

NOTE

The above guarantee avoids a potential memory corruption in scenarios where host writes to a mapped memory object have not been flushed before the memory is unmapped (or freed), and the virtual address range is subsequently reused for a different mapping (or memory allocation).

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkFlushMappedMemoryRanges` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkFlushMappedMemoryRanges-device-parameter `device` **must** be a valid `VkDevice` handle
- VUID-vkFlushMappedMemoryRanges-pMemoryRanges-parameter `pMemoryRanges` **must** be a valid pointer to an array of `memoryRangeCount` valid `VkMappedMemoryRange` structures
- VUID-vkFlushMappedMemoryRanges-memoryRangeCount-arraylength `memoryRangeCount` **must** be greater than `0`

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To invalidate ranges of non-coherent memory from the host caches, call:

```
// Provided by VK_VERSION_1_0
VkResult vkInvalidateMappedMemoryRanges(
    VkDevice          device,
    uint32_t          memoryRangeCount,
    const VkMappedMemoryRange* pMemoryRanges);
```

- `device` is the logical device that owns the memory ranges.
- `memoryRangeCount` is the length of the `pMemoryRanges` array.

- `pMemoryRanges` is a pointer to an array of `VkMappedMemoryRange` structures describing the memory ranges to invalidate.

`vkInvalidateMappedMemoryRanges` guarantees that device writes to the memory ranges described by `pMemoryRanges`, which have been made available to the host memory domain using the `VK_ACCESS_HOST_WRITE_BIT` and `VK_ACCESS_HOST_READ_BIT` access types, are made visible to the host. If a range of non-coherent memory is written by the host and then invalidated without first being flushed, its contents are undefined.

The first `synchronization scope` includes all host operations that happened-before it, as defined by the host memory model.

NOTE	This function does not synchronize with device operations directly - other host <code>synchronization operations</code> that depend on device operations such as <code>vkWaitForFences</code> must be executed beforehand. So for any non-coherent device write to be made visible to the host, there has to be a dependency chain along the following lines: 1. Device write 2. Device memory barrier including host reads in its second scope 3. Signal on the device (e.g. a <code>fence signal operation</code>) 4. Wait on the host (e.g. <code>vkWaitForFences</code>) 5. <code>vkInvalidateMappedMemoryRanges</code>

The second `synchronization scope` includes all host operations that happen-after it, as defined by the host memory model.

The first `access scope` is empty.

The second `access scope` includes host reads to the specified memory ranges.

NOTE	When a device write to a memory location is made visible to the host in this way, each whole aligned set of <code>nonCoherentAtomSize</code> bytes that the memory location exists in will also be made visible as if they were written by the device. For example, with a <code>nonCoherentAtomSize</code> of 128, if an application writes to the first byte of a memory object on the device, the first 128 bytes of the memory object will be made visible by this command. While the value of the following 127 bytes will be unchanged, this does count as an access for the purpose of synchronization, so care must be taken to avoid data races.
NOTE	Mapping non-coherent memory does not implicitly invalidate that memory.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkInvalidateMappedMemoryRanges` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkInvalidateMappedMemoryRanges-device-parameter **device** must be a valid [VkDevice](#) handle
- VUID-vkInvalidateMappedMemoryRanges-pMemoryRanges-parameter **pMemoryRanges** must be a valid pointer to an array of [memoryRangeCount](#) valid [VkMappedMemoryRange](#) structures
- VUID-vkInvalidateMappedMemoryRanges-memoryRangeCount-arraylength **memoryRangeCount** must be greater than 0

Return Codes

Success

- [VK_SUCCESS](#)

Failure

- [VK_ERROR_OUT_OF_DEVICE_MEMORY](#)
- [VK_ERROR_OUT_OF_HOST_MEMORY](#)
- [VK_ERROR_UNKNOWN](#)
- [VK_ERROR_VALIDATION_FAILED](#)

The [VkMappedMemoryRange](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkMappedMemoryRange {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceMemory      memory;
    VkDeviceSize        offset;
    VkDeviceSize        size;
} VkMappedMemoryRange;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is [NULL](#) or a pointer to a structure extending this structure.
- **memory** is the memory object to which this range belongs.
- **offset** is the zero-based byte offset from the beginning of the memory object.
- **size** is either the size of range, or [VK_WHOLE_SIZE](#) to affect the range from **offset** to the end of the current mapping of the allocation.

Valid Usage

- VUID-VkMappedMemoryRange-memory-00684
memory must be currently host mapped
- VUID-VkMappedMemoryRange-size-00685
If **size** is not equal to **VK_WHOLE_SIZE**, **offset** and **size must** specify a range contained within the currently mapped range of **memory**
- VUID-VkMappedMemoryRange-size-00686
If **size** is equal to **VK_WHOLE_SIZE**, **offset must** be within the currently mapped range of **memory**
- VUID-VkMappedMemoryRange-offset-00687
offset must be a multiple of **VkPhysicalDeviceLimits::nonCoherentAtomSize**
- VUID-VkMappedMemoryRange-size-01389
If **size** is equal to **VK_WHOLE_SIZE**, the end of the current mapping of **memory must** either be a multiple of **VkPhysicalDeviceLimits::nonCoherentAtomSize** bytes from the beginning of the memory object, or be equal to the end of the memory object
- VUID-VkMappedMemoryRange-size-01390
If **size** is not equal to **VK_WHOLE_SIZE**, **size must** either be a multiple of **VkPhysicalDeviceLimits::nonCoherentAtomSize**, or **offset** plus **size must** equal the size of **memory**

Valid Usage (Implicit)

- VUID-VkMappedMemoryRange-sType-sType
sType must be **VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE**
- VUID-VkMappedMemoryRange-pNext-pNext
pNext must be **NULL**
- VUID-VkMappedMemoryRange-memory-parameter
memory must be a valid **VkDeviceMemory** handle

To unmap a memory object once host access to it is no longer needed by the application, call:

```
// Provided by VK_VERSION_1_0
void vkUnmapMemory(
    VkDevice          device,
    VkDeviceMemory    memory);
```

- **device** is the logical device that owns the memory.
- **memory** is the memory object to be unmapped.

Valid Usage

- VUID-vkUnmapMemory-memory-00689
memory must be currently host mapped

Valid Usage (Implicit)

- VUID-vkUnmapMemory-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkUnmapMemory-memory-parameter
memory **must** be a valid [VkDeviceMemory](#) handle
- VUID-vkUnmapMemory-memory-parent
memory **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to **memory** **must** be externally synchronized

11.2.11. Lazily Allocated Memory

If the memory object is allocated from a heap with the [VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT](#) bit set, that object's backing memory **may** be provided by the implementation lazily. The actual committed size of the memory **may** initially be as small as zero (or as large as the requested size), and monotonically increases as additional memory is needed.

A memory type with this flag set is only allowed to be bound to a [VkImage](#) whose usage flags include [VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT](#).

NOTE

Using lazily allocated memory objects for framebuffer attachments that are not needed once a render pass instance has completed **may** allow some implementations to never allocate memory for such attachments.

To determine the amount of lazily-allocated memory that is currently committed for a memory object, call:

```
// Provided by VK_VERSION_1_0
void vkGetDeviceMemoryCommitment(
    VkDevice          device,
    VkDeviceMemory    memory,
    VkDeviceSize*     pCommittedMemoryInBytes);
```

- **device** is the logical device that owns the memory.
- **memory** is the memory object being queried.
- **pCommittedMemoryInBytes** is a pointer to a [VkDeviceSize](#) value in which the number of bytes currently committed is returned, on success.

The implementation **may** update the commitment at any time, and the value returned by this query **may** be out of date.

The implementation guarantees to allocate any committed memory from the `heapIndex` indicated by the memory type that the memory object was created with.

Valid Usage

- VUID-vkGetDeviceMemoryCommitment-memory-00690
`memory` **must** have been created with a memory type that reports `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT`

Valid Usage (Implicit)

- VUID-vkGetDeviceMemoryCommitment-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetDeviceMemoryCommitment-memory-parameter
`memory` **must** be a valid `VkDeviceMemory` handle
- VUID-vkGetDeviceMemoryCommitment-pCommittedMemoryInBytes-parameter
`pCommittedMemoryInBytes` **must** be a valid pointer to a `VkDeviceSize` value
- VUID-vkGetDeviceMemoryCommitment-memory-parent
`memory` **must** have been created, allocated, or retrieved from `device`

11.2.12. Protected Memory

Protected memory divides device memory into protected device memory and unprotected device memory.

Protected memory adds the following concepts:

- Memory:
 - Unprotected device memory, which **can** be visible to the device and **can** be visible to the host
 - Protected device memory, which **can** be visible to the device but **must** not be visible to the host
- Resources:
 - Unprotected images and unprotected buffers, to which unprotected memory **can** be bound
 - Protected images and protected buffers, to which protected memory **can** be bound
- Command buffers:
 - Unprotected command buffers, which **can** be submitted to a device queue to execute unprotected queue operations
 - Protected command buffers, which **can** be submitted to a protected-capable device queue to execute protected queue operations
- Device queues:

- Unprotected device queues, to which unprotected command buffers **can** be submitted
- Protected-capable device queues, to which unprotected command buffers or protected command buffers **can** be submitted
- Queue submissions
 - Unprotected queue submissions, through which unprotected command buffers **can** be submitted
 - Protected queue submissions, through which protected command buffers **can** be submitted
- Queue operations
 - Unprotected queue operations
 - Protected queue operations

Protected Memory Access Rules

If `VkPhysicalDeviceProtectedMemoryProperties::protectedNoFault` is `VK_FALSE`, applications **must** not perform any of the following operations:

- Write to unprotected memory within protected queue operations.
- Access protected memory within protected queue operations other than in framebuffer-space pipeline stages, the compute shader stage, or the transfer stage.
- Perform a query within protected queue operations.

If `VkPhysicalDeviceProtectedMemoryProperties::protectedNoFault` is `VK_TRUE`, these operations are valid, but reads will return undefined values, and writes will either be dropped or store undefined values.

Additionally, indirect operations **must** not be performed within protected queue operations.

Whether these operations are valid or not, or if any other invalid usage is performed, the implementation **must** guarantee that:

- Protected device memory **must** never be visible to the host.
- Values written to unprotected device memory **must** not be a function of values from protected memory.

11.2.13. External Memory Handle Types

QNX Screen Buffer

The QNX SDP defines `_screen_buffer` objects, which represent a buffer that the QNX Screen graphics subsystem can use directly in its windowing system APIs. More specifically, a Screen buffer is an area of memory that stores pixel data. It can be attached to Screen windows, streams, or pixmaps. These QNX Screen buffer objects **may** be imported into `VkDeviceMemory` objects for access via Vulkan. An `VkImage` or `VkBuffer` **can** be bound to the imported `VkDeviceMemory` object if it is created with `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`.

`struct _screen_buffer` is strongly typed, so naming the handle type is redundant. The internal layout

and therefore size of a `struct _screen_buffer` image may depend on native usage flags that do not have corresponding Vulkan counterparts.

QNX Screen Buffer Validity

The design of Screen in the QNX SDP makes it difficult to determine the validity of objects from outside of Screen. Therefore, applications **must** ensure that QNX Screen buffer objects provided used in various Vulkan interfaces are ones created explicitly with QNX Screen APIs. See QNX SDP documentation for more information.

A `VkDeviceMemory` imported from a QNX Screen buffer has no way to acquire a reference to its `_screen_buffer` object. Therefore, during the host execution of a Vulkan command that has a QNX Screen buffer as a parameter (including indirect parameters via `pNext` chains), the application **must** ensure that the QNX Screen buffer resource remains valid.

Generally, for a `_screen_buffer` object to be valid for use within a Vulkan implementation, the buffer object **should** have a `_screen_buffer::SCREEN_PROPERTY_USAGE` that includes at least one of: `SCREEN_USAGE_VULKAN`, `SCREEN_USAGE_OPENGL_ES2`, `SCREEN_USAGE_OPENGL_ES3`, or `SCREEN_USAGE_NATIVE`. The exact Screen-native usage flags required depends on the Vulkan implementation, and QNX Screen itself will not necessarily enforce these requirements. Note that Screen-native usage flags are in no way related to usage flags in the Vulkan specification.

QNX Screen Buffer External Formats

QNX Screen buffers **may** represent images using implementation-specific formats, layouts, color models, etc., which do not have Vulkan equivalents. Such *external formats* are commonly used by external image sources such as video decoders or cameras. Vulkan **can** import QNX Screen buffers that have external formats, but since the image contents are in an undiscoverable and possibly proprietary representation, images with external formats **must** only be used as sampled images, **must** only be sampled with a sampler that has Y'C_BC_R conversion enabled, and **must** have optimal tiling.

Images that will be backed by a QNX Screen buffer **can** use an external format by setting `VkImageCreateInfo::format` to `VK_FORMAT_UNDEFINED` and including a `VkExternalFormatQNX` structure in the `pNext` chain. Images **can** be created with an external format even if the QNX Screen buffer has a format which has an *equivalent Vulkan format* to enable consistent handling of images from sources that might use either category of format. The external format of a QNX Screen buffer **can** be obtained by passing a `VkScreenBufferFormatPropertiesQNX` structure to `vkGetScreenBufferPropertiesQNX`.

QNX Screen Buffer Image Resources

QNX Screen buffers have intrinsic width, height, format, and usage properties, so Vulkan images bound to memory imported from a QNX Screen buffer **must** use dedicated allocations: `VkMemoryDedicatedRequirements::requiresDedicatedAllocation` **must** be `VK_TRUE` for images created with `VkExternalMemoryImageCreateInfo::handleTypes` that includes `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`. When creating an image that will be bound to an imported QNX Screen buffer, the image creation parameters **must** be equivalent to the `_screen_buffer` properties as described by the valid usage of `VkMemoryAllocateInfo`.

Table 12. QNX Screen Buffer Format Equivalence

QNX Screen Format	Vulkan Format
SCREEN_FORMAT_RGBA8888	VK_FORMAT_B8G8R8A8_UNORM
SCREEN_FORMAT_RGBX8888 ¹	VK_FORMAT_B8G8R8A8_UNORM
SCREEN_FORMAT_BGRA8888	VK_FORMAT_R8G8B8A8_UNORM
SCREEN_FORMAT_BGRX8888 ¹	VK_FORMAT_R8G8B8A8_UNORM
SCREEN_FORMAT_RGBA1010102	VK_FORMAT_A2R10G10B10_UNORM_PACK32
SCREEN_FORMAT_RGBX1010102 ¹	VK_FORMAT_A2R10G10B10_UNORM_PACK32
SCREEN_FORMAT_BGRA1010102	VK_FORMAT_A2B10G10R10_UNORM_PACK32
SCREEN_FORMAT_BGRX1010102 ¹	VK_FORMAT_A2B10G10R10_UNORM_PACK32
SCREEN_FORMAT_RGBA5551	VK_FORMAT_A1R5G5B5_UNORM_PACK16
SCREEN_FORMAT_RGBX5551 ¹	VK_FORMAT_A1R5G5B5_UNORM_PACK16
SCREEN_FORMAT_RGB565	VK_FORMAT_R5G6B5_UNORM_PACK16
SCREEN_FORMAT_RGB888	VK_FORMAT_R8G8B8_UNORM

1

Vulkan does not differentiate between `SCREEN_FORMAT_RGBA8888` and `SCREEN_FORMAT_RGBX8888`: they both behave as `VK_FORMAT_R8G8B8A8_UNORM`. After an external entity writes to a `SCREEN_FORMAT_RGBX8888` QNX Screen buffer, the values read by Vulkan from the X/A component are undefined. To emulate the traditional behavior of the X component during sampling or blending, applications **should** use `VK_COMPONENT_SWIZZLE_ONE` in image view component mappings and `VK_BLEND_FACTOR_ONE` in color blend factors. There is no way to avoid copying these undefined values when copying from such an image to another image or buffer. The same behavior applies to the following pairs: `SCREEN_FORMAT_BGRA8888` and `SCREEN_FORMAT_BGRX8888`, `SCREEN_FORMAT_RGBA1010102` and `SCREEN_FORMAT_RGBX1010102`, `SCREEN_FORMAT_BGRA1010102` and `SCREEN_FORMAT_BGRX1010102`, `SCREEN_FORMAT_RGBA5551` and `SCREEN_FORMAT_RGBX5551`

11.2.14. Peer Memory Features

Peer memory is memory that is allocated for a given physical device and then bound to a resource and accessed by a different physical device, in a logical device that represents multiple physical devices. Some ways of reading and writing peer memory **may** not be supported by a device.

To determine how peer memory **can** be accessed, call:

```
// Provided by VK_VERSION_1_1
void vkGetDeviceGroupPeerMemoryFeatures(
    VkDevice          device,
    uint32_t          heapIndex,
    uint32_t          localDeviceIndex,
    uint32_t          remoteDeviceIndex,
    VkPeerMemoryFeatureFlags* pPeerMemoryFeatures);
```

- `device` is the logical device that owns the memory.

- `heapIndex` is the index of the memory heap from which the memory is allocated.
- `localDeviceIndex` is the device index of the physical device that performs the memory access.
- `remoteDeviceIndex` is the device index of the physical device that the memory is allocated for.
- `pPeerMemoryFeatures` is a pointer to a `VkPeerMemoryFeatureFlags` bitmask indicating which types of memory accesses are supported for the combination of heap, local, and remote devices.

Valid Usage

- VUID-vkGetDeviceGroupPeerMemoryFeatures-heapIndex-00691
`heapIndex` **must** be less than `memoryHeapCount`
- VUID-vkGetDeviceGroupPeerMemoryFeatures-localDeviceIndex-00692
`localDeviceIndex` **must** be a valid device index
- VUID-vkGetDeviceGroupPeerMemoryFeatures-remoteDeviceIndex-00693
`remoteDeviceIndex` **must** be a valid device index
- VUID-vkGetDeviceGroupPeerMemoryFeatures-localDeviceIndex-00694
`localDeviceIndex` **must** not equal `remoteDeviceIndex`

Valid Usage (Implicit)

- VUID-vkGetDeviceGroupPeerMemoryFeatures-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetDeviceGroupPeerMemoryFeatures-pPeerMemoryFeatures-parameter
`pPeerMemoryFeatures` **must** be a valid pointer to a `VkPeerMemoryFeatureFlags` value

Bits which **may** be set in `vkGetDeviceGroupPeerMemoryFeatures::pPeerMemoryFeatures`, indicating supported peer memory features, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkPeerMemoryFeatureFlagBits {
    VK_PEER_MEMORY_FEATURE_COPY_SRC_BIT = 0x00000001,
    VK_PEER_MEMORY_FEATURE_COPY_DST_BIT = 0x00000002,
    VK_PEER_MEMORY_FEATURE_GENERIC_SRC_BIT = 0x00000004,
    VK_PEER_MEMORY_FEATURE_GENERIC_DST_BIT = 0x00000008,
} VkPeerMemoryFeatureFlagBits;
```

- `VK_PEER_MEMORY_FEATURE_COPY_SRC_BIT` specifies that the memory **can** be accessed as the source of any `vkCmdCopy*` command.
- `VK_PEER_MEMORY_FEATURE_COPY_DST_BIT` specifies that the memory **can** be accessed as the destination of any `vkCmdCopy*` command.
- `VK_PEER_MEMORY_FEATURE_GENERIC_SRC_BIT` specifies that the memory **can** be read as any memory access type.

- `VK_PEER_MEMORY_FEATURE_GENERIC_DST_BIT` specifies that the memory **can** be written as any memory access type. Shader atomics are considered to be writes.

NOTE

The peer memory features of a memory heap also apply to any accesses that **may** be performed during [image layout transitions](#).

`VK_PEER_MEMORY_FEATURE_COPY_DST_BIT` **must** be supported for all host local heaps and for at least one device-local memory heap.

If a device does not support a peer memory feature, it is still valid to use a resource that includes both local and peer memory bindings with the corresponding access type as long as only the local bindings are actually accessed. For example, an application doing split-frame rendering would use framebuffer attachments that include both local and peer memory bindings, but would scissor the rendering to only update local memory.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkPeerMemoryFeatureFlags;
```

`VkPeerMemoryFeatureFlags` is a bitmask type for setting a mask of zero or more [VkPeerMemoryFeatureFlagBits](#).

11.2.15. Opaque Capture Address Query

To query a 64-bit opaque capture address value from a memory object, call:

```
// Provided by VK_VERSION_1_2
uint64_t vkGetDeviceMemoryOpaqueCaptureAddress(
    VkDevice device,
    const VkDeviceMemoryOpaqueCaptureAddressInfo* pInfo);
```

- `device` is the logical device that the memory object was allocated on.
- `pInfo` is a pointer to a [VkDeviceMemoryOpaqueCaptureAddressInfo](#) structure specifying the memory object to retrieve an address for.

The 64-bit return value is an opaque address representing the start of `pInfo->memory`.

If the memory object was allocated with a non-zero value of [VkMemoryOpaqueCaptureAddressAllocateInfo::opaqueCaptureAddress](#), the return value **must** be the same address.

NOTE

The expected usage for these opaque addresses is only for trace capture/replay tools to store these addresses in a trace and subsequently specify them during replay.

Valid Usage

- VUID-vkGetDeviceMemoryOpaqueCaptureAddress-None-03334

The `bufferDeviceAddress` and `bufferDeviceAddressCaptureReplay` features **must** be enabled

- VUID-vkGetDeviceMemoryOpaqueCaptureAddress-pInfo-10727
`pInfo->memory` **must** have been allocated using the `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT` flag
- VUID-vkGetDeviceMemoryOpaqueCaptureAddress-device-03335
If `device` was created with multiple physical devices, then the `bufferDeviceAddressMultiDevice` feature **must** be enabled

Valid Usage (Implicit)

- VUID-vkGetDeviceMemoryOpaqueCaptureAddress-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetDeviceMemoryOpaqueCaptureAddress-pInfo-parameter
`pInfo` **must** be a valid pointer to a valid `VkDeviceMemoryOpaqueCaptureAddressInfo` structure

The `VkDeviceMemoryOpaqueCaptureAddressInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkDeviceMemoryOpaqueCaptureAddressInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceMemory     memory;
} VkDeviceMemoryOpaqueCaptureAddressInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memory` specifies the memory whose address is being queried.

Valid Usage

- VUID-VkDeviceMemoryOpaqueCaptureAddressInfo-memory-03336
`memory` **must** have been allocated with `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT`

Valid Usage (Implicit)

- VUID-VkDeviceMemoryOpaqueCaptureAddressInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_MEMORY_OPAQUE_CAPTURE_ADDRESS_INFO`
- VUID-VkDeviceMemoryOpaqueCaptureAddressInfo-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkDeviceMemoryOpaqueCaptureAddressInfo-memory-parameter

`memory` **must** be a valid `VkDeviceMemory` handle

Chapter 12. Resource Creation

Vulkan supports two primary resource types: *buffers* and *images*. Resources are views of memory with associated formatting and dimensionality. Buffers provide access to raw arrays of bytes, whereas images **can** be multidimensional and **may** have associated metadata.

12.1. Buffers

Buffers represent linear arrays of data which are used for various purposes by binding them to a graphics or compute pipeline via descriptor sets or certain commands, or by directly specifying them as parameters to certain commands.

Buffers are represented by `VkBuffer` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkBuffer)
```

To create buffers, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateBuffer(
    VkDevice                                device,
    const VkBufferCreateInfo*               pCreateInfo,
    const VkAllocationCallbacks*           pAllocator,
    VkBuffer*                               pBuffer);
```

- `device` is the logical device that creates the buffer object.
- `pCreateInfo` is a pointer to a `VkBufferCreateInfo` structure containing parameters affecting creation of the buffer.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pBuffer` is a pointer to a `VkBuffer` handle in which the resulting buffer object is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateBuffer` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateBuffer-device-09664
`device` **must** support at least one queue family with one of the `VK_QUEUE_TRANSFER_BIT`, `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` capabilities
- VUID-vkCreateBuffer-device-05068
The number of buffers currently allocated from `device` plus 1 **must** be less than or equal to the total number of buffers requested via `VkDeviceObjectReservationCreateInfo::bufferRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateBuffer-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkCreateBuffer-pCreateInfo-parameter
`pCreateInfo` **must** be a valid pointer to a valid `VkBufferCreateInfo` structure
- VUID-vkCreateBuffer-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateBuffer-pBuffer-parameter
`pBuffer` **must** be a valid pointer to a `VkBuffer` handle
- VUID-vkCreateBuffer-device-queuecount
The device **must** have been created with at least `1` queue

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkBufferCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBufferCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkBufferCreateFlags flags;
    VkDeviceSize        size;
    VkBufferUsageFlags  usage;
    VkSharingMode        sharingMode;
    uint32_t            queueFamilyIndexCount;
    const uint32_t*      pQueueFamilyIndices;
} VkBufferCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkBufferCreateFlagBits` specifying additional parameters of the buffer.
- `size` is the size in bytes of the buffer to be created.

- **usage** is a bitmask of [VkBufferUsageFlagBits](#) specifying allowed usages of the buffer.
- **sharingMode** is a [VkSharingMode](#) value specifying the sharing mode of the buffer when it will be accessed by multiple queue families.
- **queueFamilyIndexCount** is the number of entries in the **pQueueFamilyIndices** array.
- **pQueueFamilyIndices** is a pointer to an array of queue families that will access this buffer. It is ignored if **sharingMode** is not `VK_SHARING_MODE_CONCURRENT`.

Valid Usage

- VUID-VkBufferCreateInfo-None-09499
usage **must** be a valid combination of [VkBufferUsageFlagBits](#) values
- VUID-VkBufferCreateInfo-None-09500
usage **must** not be 0
- VUID-VkBufferCreateInfo-size-00912
size **must** be greater than 0
- VUID-VkBufferCreateInfo-sharingMode-00913
If **sharingMode** is `VK_SHARING_MODE_CONCURRENT`, **pQueueFamilyIndices** **must** be a valid pointer to an array of **queueFamilyIndexCount** `uint32_t` values
- VUID-VkBufferCreateInfo-sharingMode-00914
If **sharingMode** is `VK_SHARING_MODE_CONCURRENT`, **queueFamilyIndexCount** **must** be greater than 1
- VUID-VkBufferCreateInfo-sharingMode-01419
If **sharingMode** is `VK_SHARING_MODE_CONCURRENT`, each element of **pQueueFamilyIndices** **must** be unique and **must** be less than **pQueueFamilyPropertyCount** returned by either [vkGetPhysicalDeviceQueueFamilyProperties2](#) or [vkGetPhysicalDeviceQueueFamilyProperties](#) for the **physicalDevice** that was used to create **device**
- VUID-VkBufferCreateInfo-flags-00915
flags **must** not contain `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`
- VUID-VkBufferCreateInfo-flags-00916
flags **must** not contain `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT`
- VUID-VkBufferCreateInfo-flags-00917
flags **must** not contain `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT`
- VUID-VkBufferCreateInfo-pNext-00920
If the **pNext** chain includes a [VkExternalMemoryBufferCreateInfo](#) structure, its **handleTypes** member **must** only contain bits that are also in [VkExternalBufferProperties::externalMemoryProperties.compatibleHandleTypes](#), as returned by [vkGetPhysicalDeviceExternalBufferProperties](#) with **pExternalBufferInfo->handleType** equal to any one of the handle types specified in [VkExternalMemoryBufferCreateInfo::handleTypes](#)
- VUID-VkBufferCreateInfo-flags-01887
If the **protectedMemory** feature is not enabled, **flags** **must** not contain

VK_BUFFER_CREATE_PROTECTED_BIT

- VUID-VkBufferCreateInfo-opaqueCaptureAddress-03337
If `VkBufferOpaqueCaptureAddressCreateInfo::opaqueCaptureAddress` is not zero, `flags` **must** include `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`
- VUID-VkBufferCreateInfo-flags-03338
If `flags` includes `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`, the `bufferDeviceAddressCaptureReplay` feature **must** be enabled
- VUID-VkBufferCreateInfo-flags-09641
If `flags` includes `VK_BUFFER_CREATE_PROTECTED_BIT`, then `usage` **must** not contain any of the following bits

Valid Usage (Implicit)

- VUID-VkBufferCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO`
- VUID-VkBufferCreateInfo-pNext-pNext
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkBufferOpaqueCaptureAddressCreateInfo` or `VkExternalMemoryBufferCreateInfo`
- VUID-VkBufferCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkBufferCreateInfo-flags-parameter
`flags` **must** be a valid combination of `VkBufferCreateFlagBits` values
- VUID-VkBufferCreateInfo-sharingMode-parameter
`sharingMode` **must** be a valid `VkSharingMode` value

Bits which **can** be set in `VkBufferCreateInfo::usage`, specifying usage behavior of a buffer, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkBufferUsageFlagBits {
    VK_BUFFER_USAGE_TRANSFER_SRC_BIT = 0x00000001,
    VK_BUFFER_USAGE_TRANSFER_DST_BIT = 0x00000002,
    VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT = 0x00000004,
    VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT = 0x00000008,
    VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT = 0x00000010,
    VK_BUFFER_USAGE_STORAGE_BUFFER_BIT = 0x00000020,
    VK_BUFFER_USAGE_INDEX_BUFFER_BIT = 0x00000040,
    VK_BUFFER_USAGE_VERTEX_BUFFER_BIT = 0x00000080,
    VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT = 0x00000100,
// Provided by VK_VERSION_1_2
    VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT = 0x00020000,
} VkBufferUsageFlagBits;
```

- `VK_BUFFER_USAGE_TRANSFER_SRC_BIT` specifies that the buffer **can** be used as the source of a *transfer command* (see the definition of `VK_PIPELINE_STAGE_TRANSFER_BIT`).
- `VK_BUFFER_USAGE_TRANSFER_DST_BIT` specifies that the buffer **can** be used as the destination of a transfer command.
- `VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT` specifies that the buffer **can** be used to create a `VkBufferView` suitable for occupying a `VkDescriptorSet` slot of type `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER`.
- `VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT` specifies that the buffer **can** be used to create a `VkBufferView` suitable for occupying a `VkDescriptorSet` slot of type `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`.
- `VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT` specifies that the buffer **can** be used in a `VkDescriptorBufferInfo` suitable for occupying a `VkDescriptorSet` slot either of type `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`.
- `VK_BUFFER_USAGE_STORAGE_BUFFER_BIT` specifies that the buffer **can** be used in a `VkDescriptorBufferInfo` suitable for occupying a `VkDescriptorSet` slot either of type `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`.
- `VK_BUFFER_USAGE_INDEX_BUFFER_BIT` specifies that the buffer is suitable for passing as the *buffer* parameter to `vkCmdBindIndexBuffer`.
- `VK_BUFFER_USAGE_VERTEX_BUFFER_BIT` specifies that the buffer is suitable for passing as an element of the *pBuffers* array to `vkCmdBindVertexBuffers`.
- `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` specifies that the buffer is suitable for passing as the *buffer* parameter to `vkCmdDrawIndirect`, `vkCmdDrawIndexedIndirect`, or `vkCmdDispatchIndirect`.
- `VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT` specifies that the buffer **can** be used to retrieve a buffer device address via `vkGetBufferDeviceAddress` and use that address to access the buffer's memory from a shader.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkBufferUsageFlags;
```

`VkBufferUsageFlags` is a bitmask type for setting a mask of zero or more `VkBufferUsageFlagBits`.

Bits which **can** be set in `VkBufferCreateInfo::flags`, specifying additional parameters of a buffer, are:


```
// Provided by VK_VERSION_1_0
typedef enum VkBufferCreateFlagBits {
    VK_BUFFER_CREATE_SPARSE_BINDING_BIT = 0x00000001,
    VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT = 0x00000002,
    VK_BUFFER_CREATE_SPARSE_ALIASED_BIT = 0x00000004,
    // Provided by VK_VERSION_1_1
    VK_BUFFER_CREATE_PROTECTED_BIT = 0x00000008,
    // Provided by VK_VERSION_1_2
    VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT = 0x00000010,
} VkBufferCreateFlagBits;
```

- **VK_BUFFER_CREATE_SPARSE_BINDING_BIT** specifies that the buffer will be backed using sparse memory binding. This flag is not supported in Vulkan SC [\[SCID-8\]](#).
- **VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT** specifies that the buffer **can** be partially backed using sparse memory binding. Buffers created with this flag **must** also be created with the **VK_BUFFER_CREATE_SPARSE_BINDING_BIT** flag. This flag is not supported in Vulkan SC [\[SCID-8\]](#).
- **VK_BUFFER_CREATE_SPARSE_ALIASED_BIT** specifies that the buffer will be backed using sparse memory binding with memory ranges that might also simultaneously be backing another buffer (or another portion of the same buffer). Buffers created with this flag **must** also be created with the **VK_BUFFER_CREATE_SPARSE_BINDING_BIT** flag. This flag is not supported in Vulkan SC [\[SCID-8\]](#).
- **VK_BUFFER_CREATE_PROTECTED_BIT** specifies that the buffer is a protected buffer.
- **VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT** specifies that the buffer's address **can** be saved and reused on a subsequent run (e.g. for trace capture and replay), see [VkBufferOpaqueCaptureAddressCreateInfo](#) for more detail.

See [Sparse Resource Features](#) and [Physical Device Features](#) for details of the sparse memory features supported on a device.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkBufferCreateFlags;
```

VkBufferCreateFlags is a bitmask type for setting a mask of zero or more [VkBufferCreateFlagBits](#).

To define a set of external memory handle types that **may** be used as backing store for a buffer, add a [VkExternalMemoryBufferCreateInfo](#) structure to the **pNext** chain of the [VkBufferCreateInfo](#) structure. The [VkExternalMemoryBufferCreateInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalMemoryBufferCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkExternalMemoryHandleTypeFlags handleTypes;
} VkExternalMemoryBufferCreateInfo;
```

NOTE

A `VkExternalMemoryBufferCreateInfo` structure with a non-zero `handleTypes` field must be included in the creation parameters for a buffer that will be bound to memory that is either exported or imported.

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleTypes` is zero or a bitmask of `VkExternalMemoryHandleTypeFlagBits` specifying one or more external memory handle types.

Valid Usage (Implicit)

- VUID-VkExternalMemoryBufferCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_BUFFER_CREATE_INFO`
- VUID-VkExternalMemoryBufferCreateInfo-handleTypes-parameter
`handleTypes` **must** be a valid combination of `VkExternalMemoryHandleTypeFlagBits` values

To request a specific device address for a buffer, add a `VkBufferOpaqueCaptureAddressCreateInfo` structure to the `pNext` chain of the `VkBufferCreateInfo` structure. The `VkBufferOpaqueCaptureAddressCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkBufferOpaqueCaptureAddressCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint64_t           opaqueCaptureAddress;
} VkBufferOpaqueCaptureAddressCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `opaqueCaptureAddress` is the opaque capture address requested for the buffer.

If `opaqueCaptureAddress` is zero, no specific address is requested.

If `opaqueCaptureAddress` is not zero, then it **should** be an address retrieved from `vkGetBufferOpaqueCaptureAddress` for an identically created buffer on the same implementation.

If this structure is not present, it is as if `opaqueCaptureAddress` is zero.

Applications **should** avoid creating buffers with application-provided addresses and implementation-provided addresses in the same process, to reduce the likelihood of `VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS` errors.

NOTE

The expected usage for this is that a trace capture/replay tool will add the `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT` flag to all buffers that use

`VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT`, and during capture will save the queried opaque device addresses in the trace. During replay, the buffers will be created specifying the original address so any address values stored in the trace data will remain valid.

Implementations are expected to separate such buffers in the GPU address space so normal allocations will avoid using these addresses. Applications and tools should avoid mixing application-provided and implementation-provided addresses for buffers created with `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`, to avoid address space allocation conflicts.

Valid Usage (Implicit)

- VUID-VkBufferOpaqueCaptureAddressCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_OPAQUE_CAPTURE_ADDRESS_CREATE_INFO`

To destroy a buffer, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyBuffer(
    VkDevice          device,
    VkBuffer          buffer,
    const VkAllocationCallbacks* pAllocator);
```

- `device` is the logical device that destroys the buffer.
- `buffer` is the buffer to destroy.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage

- VUID-vkDestroyBuffer-buffer-00922
All submitted commands that refer to `buffer`, either directly or via a `VkBufferView`, **must** have completed execution

Valid Usage (Implicit)

- VUID-vkDestroyBuffer-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkDestroyBuffer-buffer-parameter
If `buffer` is not `VK_NULL_HANDLE`, `buffer` **must** be a valid `VkBuffer` handle
- VUID-vkDestroyBuffer-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkDestroyBuffer-buffer-parent

If **buffer** is a valid handle, it **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to **buffer** **must** be externally synchronized

12.2. Buffer Views

A *buffer view* represents a contiguous range of a buffer and a specific format to be used to interpret the data. Buffer views are used to enable shaders to access buffer contents using [image operations](#). In order to create a valid buffer view, the buffer **must** have been created with at least one of the following usage flags:

- **VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT**
- **VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT**

Buffer views are represented by **VkBufferView** handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkBufferView)
```

To create a buffer view, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateBufferView(
    VkDevice                                device,
    const VkBufferViewCreateInfo*           pCreateInfo,
    const VkAllocationCallbacks*           pAllocator,
    VkBufferView*                           pView);
```

- **device** is the logical device that creates the buffer view.
- **pCreateInfo** is a pointer to a **VkBufferViewCreateInfo** structure containing parameters to be used to create the buffer view.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.
- **pView** is a pointer to a **VkBufferView** handle in which the resulting buffer view object is returned.

If **VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations** is **VK_TRUE**, **vkCreateBufferView** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage

- VUID-vkCreateBufferView-device-09665

device **must** support at least one queue family with one of the `VK_QUEUE_COMPUTE_BIT` or `VK_QUEUE_GRAPHICS_BIT` capabilities

- VUID-vkCreateBufferView-device-05068

The number of buffer views currently allocated from **device** plus 1 **must** be less than or equal to the total number of buffer views requested via `VkDeviceObjectReservationCreateInfo::bufferViewRequestCount` specified when **device** was created

Valid Usage (Implicit)

- VUID-vkCreateBufferView-device-parameter
device **must** be a valid `VkDevice` handle
- VUID-vkCreateBufferView-pCreateInfo-parameter
pCreateInfo **must** be a valid pointer to a valid `VkBufferViewCreateInfo` structure
- VUID-vkCreateBufferView-pAllocator-null
pAllocator **must** be `NULL`
- VUID-vkCreateBufferView-pView-parameter
pView **must** be a valid pointer to a `VkBufferView` handle
- VUID-vkCreateBufferView-device-queuecount
The device **must** have been created with at least 1 queue

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkBufferViewCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBufferViewCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkBufferViewCreateFlags flags;
    VkBuffer              buffer;
    VkFormat              format;
    VkDeviceSize          offset;
    VkDeviceSize          range;
} VkBufferViewCreateInfo;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **flags** is reserved for future use.
- **buffer** is a **VkBuffer** on which the view will be created.
- **format** is a **VkFormat** describing the format of the data elements in the buffer.
- **offset** is an offset in bytes from the base address of the buffer. Accesses to the buffer view from shaders use addressing that is relative to this starting offset.
- **range** is a size in bytes of the buffer view. If **range** is equal to **VK_WHOLE_SIZE**, the range from **offset** to the end of the buffer is used. If **VK_WHOLE_SIZE** is used and the remaining size of the buffer is not a multiple of the **texel block size** of **format**, the nearest smaller multiple is used.

The buffer view has a *buffer view usage* identifying which descriptor types can be created from it. This usage is equal to the **VkBufferCreateInfo::usage** value used to create **buffer**.

Valid Usage

- VUID-VkBufferViewCreateInfo-offset-00925
offset must be less than the size of **buffer**
- VUID-VkBufferViewCreateInfo-range-00928
If **range** is not equal to **VK_WHOLE_SIZE**, **range must** be greater than 0
- VUID-VkBufferViewCreateInfo-range-00929
If **range** is not equal to **VK_WHOLE_SIZE**, **range must** be an integer multiple of the texel block size of **format**
- VUID-VkBufferViewCreateInfo-range-00930
If **range** is not equal to **VK_WHOLE_SIZE**, the number of texel buffer elements given by $(\lceil \text{range} / (\text{texel block size}) \rceil \times (\text{texels per block}))$ where texel block size and texels per block are as defined in the **Compatible Formats** table for **format**, **must** be less than or equal to **VkPhysicalDeviceLimits::maxTexelBufferElements**
- VUID-VkBufferViewCreateInfo-offset-00931
If **range** is not equal to **VK_WHOLE_SIZE**, the sum of **offset** and **range must** be less than or equal to the size of **buffer**
- VUID-VkBufferViewCreateInfo-range-04059

If **range** is equal to **VK_WHOLE_SIZE**, the number of texel buffer elements given by $\lceil (\text{size} - \text{offset}) / (\text{texel block size}) \rceil \times (\text{texels per block})$ where **size** is the size of **buffer**, and texel block size and texels per block are as defined in the [Compatible Formats](#) table for **format**, **must** be less than or equal to **VkPhysicalDeviceLimits::maxTexelBufferElements**

- VUID-VkBufferViewCreateInfo-buffer-00932

buffer **must** have been created with a **usage** value containing at least one of **VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT** or **VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT**

- VUID-VkBufferViewCreateInfo-format-08778

If the **buffer view usage** contains **VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT**, then **format features** of **format** **must** contain **VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT**

- VUID-VkBufferViewCreateInfo-format-08779

If the **buffer view usage** contains **VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT**, then **format features** of **format** **must** contain **VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT**

- VUID-VkBufferViewCreateInfo-buffer-00935

If **buffer** is non-sparse then it **must** be bound completely and contiguously to a single **VkDeviceMemory** object

- VUID-VkBufferViewCreateInfo-offset-02749

If the **texelBufferAlignment** feature is not enabled, **offset** **must** be a multiple of **VkPhysicalDeviceLimits::minTexelBufferOffsetAlignment**

- VUID-VkBufferViewCreateInfo-buffer-02750

If the **texelBufferAlignment** feature is enabled and if **buffer** was created with **usage** containing **VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT**, **offset** **must** be a multiple of the lesser of **VkPhysicalDeviceTexelBufferAlignmentProperties::storageTexelBufferOffsetAlignmentBytes** or, if **VkPhysicalDeviceTexelBufferAlignmentProperties::storageTexelBufferOffsetSingleTexelAlignment** is **VK_TRUE**, the size of a texel of the requested **format**. If the size of a texel is a multiple of three bytes, then the size of a single component of **format** is used instead

- VUID-VkBufferViewCreateInfo-buffer-02751

If the **texelBufferAlignment** feature is enabled and if **buffer** was created with **usage** containing **VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT**, **offset** **must** be a multiple of the lesser of **VkPhysicalDeviceTexelBufferAlignmentProperties::uniformTexelBufferOffsetAlignmentBytes** or, if **VkPhysicalDeviceTexelBufferAlignmentProperties::uniformTexelBufferOffsetSingleTexelAlignment** is **VK_TRUE**, the size of a texel of the requested **format**. If the size of a texel is a multiple of three bytes, then the size of a single component of **format** is used instead

Valid Usage (Implicit)

- VUID-VkBufferViewCreateInfo-sType-sType

sType **must** be **VK_STRUCTURE_TYPE_BUFFER_VIEW_CREATE_INFO**

- VUID-VkBufferViewCreateInfo-pNext-pNext

pNext **must** be **NULL**

- VUID-VkBufferViewCreateInfo-flags-zeroBitmask

flags must be 0

- VUID-VkBufferViewCreateInfo-buffer-parameter
buffer must be a valid [VkBuffer](#) handle
- VUID-VkBufferViewCreateInfo-format-parameter
format must be a valid [VkFormat](#) value

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkBufferViewCreateFlags;
```

VkBufferViewCreateFlags is a bitmask type for setting a mask, but is currently reserved for future use.

To destroy a buffer view, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyBufferView(
    VkDevice          device,
    VkBufferView      bufferView,
    const VkAllocationCallbacks* pAllocator);
```

- **device** is the logical device that destroys the buffer view.
- **bufferView** is the buffer view to destroy.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage

- VUID-vkDestroyBufferView-bufferView-00936
All submitted commands that refer to **bufferView must** have completed execution

Valid Usage (Implicit)

- VUID-vkDestroyBufferView-device-parameter
device must be a valid [VkDevice](#) handle
- VUID-vkDestroyBufferView-bufferView-parameter
If **bufferView** is not [VK_NULL_HANDLE](#), **bufferView must** be a valid [VkBufferView](#) handle
- VUID-vkDestroyBufferView-pAllocator-null
pAllocator must be [NULL](#)
- VUID-vkDestroyBufferView-bufferView-parent
If **bufferView** is a valid handle, it **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to `bufferView` **must** be externally synchronized

12.2.1. Buffer View Format Features

Valid uses of a `VkBufferView` **may** depend on the buffer view's *format features*, defined below. Such constraints are documented in the affected valid usage statement.

- The buffer view's set of *format features* is the value of `VkFormatProperties::bufferFeatures` found by calling `vkGetPhysicalDeviceFormatProperties` on the same `format` as `VkBufferViewCreateInfo::format`.

12.3. Buffer Device Addresses

To query a 64-bit buffer device address value which can be used to identify a buffer to API commands or through which buffer memory **can** be accessed, call:

```
// Provided by VK_VERSION_1_2
VkDeviceAddress vkGetBufferDeviceAddress(
    VkDevice device,
    const VkBufferDeviceAddressInfo* pInfo);
```

- `device` is the logical device that the buffer was created on.
- `pInfo` is a pointer to a `VkBufferDeviceAddressInfo` structure specifying the buffer to retrieve an address for.

The 64-bit return value, `bufferBaseAddress`, is an address of the start of `pInfo->buffer`. Addresses in the range `[bufferBaseAddress, bufferBaseAddress + VkBufferCreateInfo::size)` **can** be used to access the memory bound to this buffer on the device.

A value of zero is reserved as a “null” pointer and **must** not be returned as a valid buffer device address.

If the buffer was created with a non-zero value of `VkBufferOpaqueCaptureAddressCreateInfo::opaqueCaptureAddress`, the return value will be the same address that was returned at capture time.

The returned address **must** satisfy the alignment requirement specified by `VkMemoryRequirements::alignment` for the buffer in `VkBufferDeviceAddressInfo::buffer`.

If multiple `VkBuffer` objects are bound to overlapping ranges of `VkDeviceMemory`, implementations **may** return address ranges which overlap. In this case, it is ambiguous which `VkBuffer` is associated with any given device address. For purposes of valid usage, if multiple `VkBuffer` objects **can** be attributed to a device address, a `VkBuffer` is selected such that valid usage passes, if it exists.

Valid Usage

- VUID-vkGetBufferDeviceAddress-bufferDeviceAddress-03324

The `bufferDeviceAddress` feature **must** be enabled, and at least one of the following conditions **must** be met

- `buffer` is bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkGetBufferDeviceAddress-device-03325
If `device` was created with multiple physical devices, then the `bufferDeviceAddressMultiDevice` feature **must** be enabled

Valid Usage (Implicit)

- VUID-vkGetBufferDeviceAddress-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetBufferDeviceAddress-pInfo-parameter
`pInfo` **must** be a valid pointer to a valid `VkBufferDeviceAddressInfo` structure

The `VkBufferDeviceAddressInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkBufferDeviceAddressInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkBuffer            buffer;
} VkBufferDeviceAddressInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `buffer` specifies the buffer whose address is being queried.

Valid Usage

- VUID-VkBufferDeviceAddressInfo-buffer-02601
`buffer` **must** have been created with `VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT`

Valid Usage (Implicit)

- VUID-VkBufferDeviceAddressInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_INFO`
- VUID-VkBufferDeviceAddressInfo-pNext-pNext
`pNext` **must** be `NULL`

- VUID-VkBufferDeviceAddressInfo-buffer-parameter
buffer **must** be a valid [VkBuffer](#) handle

To query a 64-bit buffer opaque capture address, call:

```
// Provided by VK_VERSION_1_2
uint64_t vkGetBufferOpaqueCaptureAddress(
    VkDevice device,
    const VkBufferDeviceAddressInfo* pInfo);
```

- **device** is the logical device that the buffer was created on.
- **pInfo** is a pointer to a [VkBufferDeviceAddressInfo](#) structure specifying the buffer to retrieve an address for.

The 64-bit return value is an opaque capture address of the start of **pInfo->buffer**.

If the buffer was created with a non-zero value of [VkBufferOpaqueCaptureAddressCreateInfo::opaqueCaptureAddress](#) the return value **must** be the same address.

Valid Usage

- VUID-vkGetBufferOpaqueCaptureAddress-None-03326
The **bufferDeviceAddress** and **bufferDeviceAddressCaptureReplay** features **must** be enabled
- VUID-vkGetBufferOpaqueCaptureAddress-pInfo-10725
pInfo->buffer **must** have been created with the **VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT** flag
- VUID-vkGetBufferOpaqueCaptureAddress-device-03327
If **device** was created with multiple physical devices, then the **bufferDeviceAddressMultiDevice** feature **must** be enabled

Valid Usage (Implicit)

- VUID-vkGetBufferOpaqueCaptureAddress-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkGetBufferOpaqueCaptureAddress-pInfo-parameter
pInfo **must** be a valid pointer to a valid [VkBufferDeviceAddressInfo](#) structure

12.4. Images

Images are specialized resources that have multi-dimensional access, as outlined in the [Images](#) chapter. Images **can** be used for various purposes, such as [rendering attachments](#), [for copy operations](#), or accessed through shaders via [resource descriptors](#).

Images are represented by `VkImage` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkImage)
```

To create images, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateImage(
    VkDevice                                device,
    const VkImageCreateInfo*                pCreateInfo,
    const VkAllocationCallbacks*            pAllocator,
    VkImage*                                pImage);
```

- `device` is the logical device that creates the image.
- `pCreateInfo` is a pointer to a `VkImageCreateInfo` structure containing parameters to be used to create the image.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pImage` is a pointer to a `VkImage` handle in which the resulting image object is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateImage` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateImage-device-09666
`device` **must** support at least one queue family with one of the `VK_QUEUE_TRANSFER_BIT`, `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` capabilities
- VUID-vkCreateImage-device-05068
The number of images currently allocated from `device` plus 1 **must** be less than or equal to the total number of images requested via `VkDeviceObjectReservationCreateInfo::imageRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateImage-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkCreateImage-pCreateInfo-parameter
`pCreateInfo` **must** be a valid pointer to a valid `VkImageCreateInfo` structure
- VUID-vkCreateImage-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateImage-pImage-parameter

`pImage` **must** be a valid pointer to a `VkImage` handle

- `VUID-vkCreateImage-device-queuecount`
The device **must** have been created with at least `1` queue

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkImageCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkImageCreateFlags    flags;
    VkImageType           imageType;
    VkFormat              format;
    VkExtent3D            extent;
    uint32_t              mipLevels;
    uint32_t              arrayLayers;
    VkSampleCountFlagBits samples;
    VkImageTiling          tiling;
    VkImageUsageFlags      usage;
    VkSharingMode          sharingMode;
    uint32_t              queueFamilyIndexCount;
    const uint32_t*        pQueueFamilyIndices;
    VkImageLayout          initialLayout;
} VkImageCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkImageCreateFlagBits` describing additional parameters of the image.
- `imageType` is a `VkImageType` value specifying the basic dimensionality of the image. Layers in array textures do not count as a dimension for the purposes of the image type.
- `format` is a `VkFormat` describing the format and type of the texel blocks that will be contained in the image.

- **extent** is a [VkExtent3D](#) describing the number of texels/pixels in each dimension of the base level.
- **mipLevels** describes the number of levels of detail available for minified sampling of the image.
- **arrayLayers** is the number of layers in the image.
- **samples** is a [VkSampleCountFlagBits](#) value specifying the number of [samples per texel](#).
- **tiling** is a [VkImageTiling](#) value specifying the tiling arrangement of the texel blocks in memory.
- **usage** is a bitmask of [VkImageUsageFlagBits](#) describing the intended usage of the image.
- **sharingMode** is a [VkSharingMode](#) value specifying the sharing mode of the image when it will be accessed by multiple queue families.
- **queueFamilyIndexCount** is the number of entries in the **pQueueFamilyIndices** array.
- **pQueueFamilyIndices** is a pointer to an array of queue families that will access this image. It is ignored if **sharingMode** is not [VK_SHARING_MODE_CONCURRENT](#).
- **initialLayout** is a [VkImageLayout](#) value specifying the initial [VkImageLayout](#) of all image subresources of the image. See [Image Layouts](#).

Images created with **tiling** equal to [VK_IMAGE_TILING_LINEAR](#) have further restrictions on their limits and capabilities compared to images created with **tiling** equal to [VK_IMAGE_TILING_OPTIMAL](#). Creation of images with tiling [VK_IMAGE_TILING_LINEAR](#) **may** not be supported unless other parameters meet all of the constraints:

- **imageType** is [VK_IMAGE_TYPE_2D](#)
- **format** is not a depth/stencil format
- **mipLevels** is 1
- **arrayLayers** is 1
- **samples** is [VK_SAMPLE_COUNT_1_BIT](#)
- **usage** only includes [VK_IMAGE_USAGE_TRANSFER_SRC_BIT](#) and/or [VK_IMAGE_USAGE_TRANSFER_DST_BIT](#)

Images created with one of the [formats that require a sampler Y'C_BC_R conversion](#), have further restrictions on their limits and capabilities compared to images created with other formats. Creation of images with a format requiring [Y'C_BC_R conversion](#) **may** not be supported unless other parameters meet all of the constraints:

- **imageType** is [VK_IMAGE_TYPE_2D](#)
- **mipLevels** is 1
- **arrayLayers** is 1, unless the **ycbcrImageArrays** feature is enabled, or otherwise indicated by [VkImageFormatProperties::maxArrayLayers](#), as returned by [vkGetPhysicalDeviceImageFormatProperties](#)
- **samples** is [VK_SAMPLE_COUNT_1_BIT](#)

Implementations **may** support additional limits and capabilities beyond those listed above.

To determine the set of valid **usage** bits for a given format, call [vkGetPhysicalDeviceFormatProperties](#).

If the size of the resultant image would exceed `maxResourceSize`, then `vkCreateImage` **must** fail and return `VK_ERROR_OUT_OF_DEVICE_MEMORY`. This failure **may** occur even when all image creation parameters satisfy their valid usage requirements.

NOTE

For images created without `VK_IMAGE_CREATE_EXTENDED_USAGE_BIT` a `usage` bit is valid if it is supported for the format the image is created with.

For images created with `VK_IMAGE_CREATE_EXTENDED_USAGE_BIT` a `usage` bit is valid if it is supported for at least one of the formats a `VkImageView` created from the image **can** have (see [Image Views](#) for more detail).

Image Creation Limits

Valid values for some image creation parameters are limited by a numerical upper bound or by inclusion in a bitset. For example, `VkImageCreateInfo::arrayLayers` is limited by `imageCreateMaxArrayLayers`, defined below; and `VkImageCreateInfo::samples` is limited by `imageCreateSampleCounts`, also defined below.

Several limiting values are defined below, as well as assisting values from which the limiting values are derived. The limiting values are referenced by the relevant valid usage statements of `VkImageCreateInfo`.

- Let `uint64_t imageCreateDrmFormatModifiers[]` be the set of [Linux DRM format modifiers](#) that the resultant image **may** have.
 - If `tiling` is not `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `imageCreateDrmFormatModifiers` is empty.
 - If `VkImageCreateInfo::pNext` contains `VkImageDrmFormatModifierExplicitCreateInfoEXT`, then `imageCreateDrmFormatModifiers` contains exactly one modifier, `VkImageDrmFormatModifierExplicitCreateInfoEXT::drmFormatModifier`.
 - If `VkImageCreateInfo::pNext` contains `VkImageDrmFormatModifierListCreateInfoEXT`, then `imageCreateDrmFormatModifiers` contains the entire array `VkImageDrmFormatModifierListCreateInfoEXT::pDrmFormatModifiers`.
- Let `VkBool32 imageCreateMaybeLinear` indicate if the resultant image may be [linear](#).
 - If `tiling` is `VK_IMAGE_TILING_LINEAR`, then `imageCreateMaybeLinear` is `VK_TRUE`.
 - If `tiling` is `VK_IMAGE_TILING_OPTIMAL`, then `imageCreateMaybeLinear` is `VK_FALSE`.
 - If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `imageCreateMaybeLinear` is `VK_TRUE` if and only if `imageCreateDrmFormatModifiers` contains `DRM_FORMAT_MOD_LINEAR`.
- Let `VkFormatFeatureFlags imageCreateFormatFeatures` be the set of valid *format features* available during image creation.
 - If `tiling` is `VK_IMAGE_TILING_LINEAR`, then `imageCreateFormatFeatures` is the value of `VkFormatProperties::linearTilingFeatures` found by calling `vkGetPhysicalDeviceFormatProperties` with parameter `format` equal to `VkImageCreateInfo::format`.

- If `tiling` is `VK_IMAGE_TILING_OPTIMAL`, and if the `pNext` chain includes no `VkExternalFormatQNX` structure with non-zero `externalFormat`, then `imageCreateFormatFeatures` is the value of `VkFormatProperties::optimalTilingFeatures` found by calling `vkGetPhysicalDeviceFormatProperties` with parameter `format` equal to `VkImageCreateInfo::format`.
 - If `tiling` is `VK_IMAGE_TILING_OPTIMAL`, and if the `pNext` chain includes a `VkExternalFormatQNX` structure with non-zero `externalFormat`, then `imageCreateFormatFeatures` is the value of `VkScreenBufferFormatPropertiesQNX::formatFeatures` obtained by `vkGetScreenBufferPropertiesQNX` with a matching `externalFormat` value.
 - If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then the value of `imageCreateFormatFeatures` is found by calling `vkGetPhysicalDeviceFormatProperties2` with `VkImageFormatProperties::format` equal to `VkImageCreateInfo::format` and with `VkDrmFormatModifierPropertiesListEXT` chained into `VkFormatProperties2`; by collecting all members of the returned array `VkDrmFormatModifierPropertiesListEXT::pDrmFormatModifierProperties` whose `drmFormatModifier` belongs to `imageCreateDrmFormatModifiers`; and by taking the bitwise intersection, over the collected array members, of `drmFormatModifierTilingFeatures`. (The resultant `imageCreateFormatFeatures` **may** be empty).
- Let `VkImageFormatProperties2 imageCreateImageFormatPropertiesList[]` be defined as follows.
 - If `VkImageCreateInfo::pNext` contains no `VkExternalFormatQNX` structure with non-zero `externalFormat`, then `imageCreateImageFormatPropertiesList` is the list of structures obtained by calling `vkGetPhysicalDeviceImageFormatProperties2`, possibly multiple times, as follows:
 - The parameters `VkPhysicalDeviceImageFormatInfo2::format`, `imageType`, `tiling`, `usage`, and `flags` **must** be equal to those in `VkImageCreateInfo`.
 - If `VkImageCreateInfo::pNext` contains a `VkExternalMemoryImageCreateInfo` structure whose `handleTypes` is not `0`, then `VkPhysicalDeviceImageFormatInfo2::pNext` **must** contain a `VkPhysicalDeviceExternalImageFormatInfo` structure whose `handleType` is not `0`; and `vkGetPhysicalDeviceImageFormatProperties2` **must** be called for each handle type in `VkExternalMemoryImageCreateInfo::handleTypes`, successively setting `VkPhysicalDeviceExternalImageFormatInfo::handleType` on each call.
 - If `VkImageCreateInfo::pNext` contains no `VkExternalMemoryImageCreateInfo` structure, or contains a structure whose `handleTypes` is `0`, then `VkPhysicalDeviceImageFormatInfo2::pNext` **must** either contain no `VkPhysicalDeviceExternalImageFormatInfo` structure, or contain a structure whose `handleType` is `0`.
 - If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then:
 - `VkPhysicalDeviceImageFormatInfo2::pNext` **must** contain a `VkPhysicalDeviceImageDrmFormatModifierInfoEXT` structure where `sharingMode` is equal to `VkImageCreateInfo::sharingMode`;
 - if `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, then `queueFamilyIndexCount` and

- `pQueueFamilyIndices` **must** be equal to those in `VkImageCreateInfo`;
- if `flags` contains `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`, then the `VkImageFormatListCreateInfo` structure included in the `pNext` chain of `VkPhysicalDeviceImageFormatInfo2` **must** be equivalent to the one included in the `pNext` chain of `VkImageCreateInfo`;
- `vkGetPhysicalDeviceImageFormatProperties2` **must** be called for each modifier in `imageCreateDrmFormatModifiers`, successively setting `VkPhysicalDeviceImageDrmFormatModifierInfoEXT::drmFormatModifier` on each call.
- If `tiling` is not `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `VkPhysicalDeviceImageFormatInfo2::pNext` **must** contain no `VkPhysicalDeviceImageDrmFormatModifierInfoEXT` structure.
- If any call to `vkGetPhysicalDeviceImageFormatProperties2` returns an error, then `imageCreateImageFormatPropertiesList` is defined to be the empty list.
- Let `uint32_t imageCreateMaxMipLevels` be the minimum value of `VkImageFormatProperties::maxMipLevels` in `imageCreateImageFormatPropertiesList`. The value is undefined if `imageCreateImageFormatPropertiesList` is empty.
- Let `uint32_t imageCreateMaxArrayLayers` be the minimum value of `VkImageFormatProperties::maxArrayLayers` in `imageCreateImageFormatPropertiesList`. The value is undefined if `imageCreateImageFormatPropertiesList` is empty.
- Let `VkExtent3D imageCreateMaxExtent` be the component-wise minimum over all `VkImageFormatProperties::maxExtent` values in `imageCreateImageFormatPropertiesList`. The value is undefined if `imageCreateImageFormatPropertiesList` is empty.
- Let `VkSampleCountFlags imageCreateSampleCounts` be the intersection of each `VkImageFormatProperties::sampleCounts` in `imageCreateImageFormatPropertiesList`. The value is undefined if `imageCreateImageFormatPropertiesList` is empty.

Valid Usage

- VUID-VkImageCreateInfo-imageCreateMaxMipLevels-02251
Each of the following values (as described in [Image Creation Limits](#)) **must** not be undefined : `imageCreateMaxMipLevels`, `imageCreateMaxArrayLayers`, `imageCreateMaxExtent`, and `imageCreateSampleCounts`
- VUID-VkImageCreateInfo-sharingMode-00941
If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, `pQueueFamilyIndices` **must** be a valid pointer to an array of `queueFamilyIndexCount` `uint32_t` values
- VUID-VkImageCreateInfo-sharingMode-00942
If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, `queueFamilyIndexCount` **must** be greater than 1
- VUID-VkImageCreateInfo-sharingMode-01420
If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, each element of `pQueueFamilyIndices` **must** be unique and **must** be less than `pQueueFamilyPropertyCount` returned by either

[vkGetPhysicalDeviceQueueFamilyProperties](#)

or

[vkGetPhysicalDeviceQueueFamilyProperties2](#) for the `physicalDevice` that was used to create `device`

- VUID-VkImageCreateInfo-format-00943
`format` **must** not be `VK_FORMAT_UNDEFINED`
- VUID-VkImageCreateInfo-extent-00944
`extent.width` **must** be greater than 0
- VUID-VkImageCreateInfo-extent-00945
`extent.height` **must** be greater than 0
- VUID-VkImageCreateInfo-extent-00946
`extent.depth` **must** be greater than 0
- VUID-VkImageCreateInfo-mipLevels-00947
`mipLevels` **must** be greater than 0
- VUID-VkImageCreateInfo-arrayLayers-00948
`arrayLayers` **must** be greater than 0
- VUID-VkImageCreateInfo-flags-00949
If `flags` contains `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`, `imageType` **must** be `VK_IMAGE_TYPE_2D`
- VUID-VkImageCreateInfo-flags-08865
If `flags` contains `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`, `extent.width` and `extent.height` **must** be equal
- VUID-VkImageCreateInfo-flags-08866
If `flags` contains `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`, `arrayLayers` **must** be greater than or equal to 6
- VUID-VkImageCreateInfo-flags-00950
If `flags` contains `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT`, `imageType` **must** be `VK_IMAGE_TYPE_3D`
- VUID-VkImageCreateInfo-flags-09403
If `flags` contains `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT`, `flags` **must** not include `VK_IMAGE_CREATE_SPARSE_ALIASED_BIT`, `VK_IMAGE_CREATE_SPARSE_BINDING_BIT`, or `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`
- VUID-VkImageCreateInfo-extent-02252
`extent.width` **must** be less than or equal to `imageCreateMaxExtent.width` (as defined in [Image Creation Limits](#))
- VUID-VkImageCreateInfo-extent-02253
`extent.height` **must** be less than or equal to `imageCreateMaxExtent.height` (as defined in [Image Creation Limits](#))
- VUID-VkImageCreateInfo-extent-02254
`extent.depth` **must** be less than or equal to `imageCreateMaxExtent.depth` (as defined in [Image Creation Limits](#))
- VUID-VkImageCreateInfo-imageType-00956
If `imageType` is `VK_IMAGE_TYPE_1D`, both `extent.height` and `extent.depth` **must** be 1

- VUID-VkImageCreateInfo-imageType-00957
If `imageType` is `VK_IMAGE_TYPE_2D`, `extent.depth` **must** be 1
- VUID-VkImageCreateInfo-mipLevels-00958
`mipLevels` **must** be less than or equal to the number of levels in the complete mipmap chain based on `extent.width`, `extent.height`, and `extent.depth`
- VUID-VkImageCreateInfo-mipLevels-02255
`mipLevels` **must** be less than or equal to `imageCreateMaxMipLevels` (as defined in [Image Creation Limits](#))
- VUID-VkImageCreateInfo-arrayLayers-02256
`arrayLayers` **must** be less than or equal to `imageCreateMaxArrayLayers` (as defined in [Image Creation Limits](#))
- VUID-VkImageCreateInfo-imageType-00961
If `imageType` is `VK_IMAGE_TYPE_3D`, `arrayLayers` **must** be 1
- VUID-VkImageCreateInfo-samples-02257
If `samples` is not `VK_SAMPLE_COUNT_1_BIT`, then `imageType` **must** be `VK_IMAGE_TYPE_2D`, `flags` **must** not contain `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`, `mipLevels` **must** be equal to 1, and `imageCreateMayBeLinear` (as defined in [Image Creation Limits](#)) **must** be `VK_FALSE`,
- VUID-VkImageCreateInfo-usage-00963
If `usage` includes `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`, then bits other than `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, and `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT` **must** not be set
- VUID-VkImageCreateInfo-usage-00964
If `usage` includes `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`, or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`, `extent.width` **must** be less than or equal to `VkPhysicalDeviceLimits::maxFramebufferWidth`
- VUID-VkImageCreateInfo-usage-00965
If `usage` includes `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`, or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`, `extent.height` **must** be less than or equal to `VkPhysicalDeviceLimits::maxFramebufferHeight`
- VUID-VkImageCreateInfo-usage-00966
If `usage` includes `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`, `usage` **must** also contain at least one of `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`, `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`
- VUID-VkImageCreateInfo-samples-02258
`samples` **must** be a valid [VkSampleCountFlagBits](#) value that is set in `imageCreateSampleCounts` (as defined in [Image Creation Limits](#))
- VUID-VkImageCreateInfo-usage-00968
If the `shaderStorageImageMultisample` feature is not enabled, and `usage` contains `VK_IMAGE_USAGE_STORAGE_BIT`, `samples` **must** be `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkImageCreateInfo-flags-05062
`flags` **must** not contain `VK_IMAGE_CREATE_SPARSE_BINDING_BIT`,

VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT, VK_IMAGE_CREATE_SPARSE_ALIASED_BIT, or
VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT

- VUID-VkImageCreateInfo-flags-01890

If the `protectedMemory` feature is not enabled, `flags` **must** not contain `VK_IMAGE_CREATE_PROTECTED_BIT`

- VUID-VkImageCreateInfo-pNext-00990

If the `pNext` chain includes a `VkExternalMemoryImageCreateInfo` structure, its `handleTypes` member **must** only contain bits that are also in `VkExternalImageFormatProperties::externalMemoryProperties.compatibleHandleTypes`, as returned by `vkGetPhysicalDeviceImageFormatProperties2` with `format`, `imageType`, `tiling`, `usage`, and `flags` equal to those in this structure, and with a `VkPhysicalDeviceExternalImageFormatInfo` structure included in the `pNext` chain, with a `handleType` equal to any one of the handle types specified in `VkExternalMemoryImageCreateInfo::handleTypes`

- VUID-VkImageCreateInfo-flags-01572

If `flags` contains `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT`, then `format` **must** be a `compressed image format`

- VUID-VkImageCreateInfo-flags-01573

If `flags` contains `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT`, then `flags` **must** also contain `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`

- VUID-VkImageCreateInfo-initialLayout-00993

`initialLayout` **must** be `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`

- VUID-VkImageCreateInfo-pNext-01443

If the `pNext` chain includes a `VkExternalMemoryImageCreateInfo` or `VkExternalMemoryImageCreateInfoNV` structure whose `handleTypes` member is not `0`, `initialLayout` **must** be `VK_IMAGE_LAYOUT_UNDEFINED`

- VUID-VkImageCreateInfo-format-06410

If the image `format` is one of the `formats that require a sampler Y'CBCR conversion`, `mipLevels` **must** be 1

- VUID-VkImageCreateInfo-format-06411

If the image `format` is one of the `formats that require a sampler Y'CBCR conversion`, `samples` **must** be `VK_SAMPLE_COUNT_1_BIT`

- VUID-VkImageCreateInfo-format-06412

If the image `format` is one of the `formats that require a sampler Y'CBCR conversion`, `imageType` **must** be `VK_IMAGE_TYPE_2D`

- VUID-VkImageCreateInfo-imageCreateFormatFeatures-02260

If `format` is a *multi-planar* format, and if `imageCreateFormatFeatures` (as defined in `Image Creation Limits`) does not contain `VK_FORMAT_FEATURE_DISJOINT_BIT`, then `flags` **must** not contain `VK_IMAGE_CREATE_DISJOINT_BIT`

- VUID-VkImageCreateInfo-format-01577

If `format` is not a *multi-planar* format, and `flags` does not include `VK_IMAGE_CREATE_ALIAS_BIT`, `flags` **must** not contain `VK_IMAGE_CREATE_DISJOINT_BIT`

- VUID-VkImageCreateInfo-format-04712

If `format` has a `_422` or `_420` suffix, `extent.width` **must** be a multiple of 2

- VUID-VkImageCreateInfo-format-04713

If `format` has a `_420` suffix, `extent.height` **must** be a multiple of 2

- VUID-VkImageCreateInfo-tiling-02261

If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then the `pNext` chain **must** include exactly one of `VkImageDrmFormatModifierListCreateInfoEXT` or `VkImageDrmFormatModifierExplicitCreateInfoEXT` structures

- VUID-VkImageCreateInfo-pNext-02262

If the `pNext` chain includes a `VkImageDrmFormatModifierListCreateInfoEXT` or `VkImageDrmFormatModifierExplicitCreateInfoEXT` structure, then `tiling` **must** be `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`

- VUID-VkImageCreateInfo-tiling-02353

If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` and `flags` contains `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`, then the `pNext` chain **must** include a `VkImageFormatListCreateInfo` structure with non-zero `viewFormatCount`

- VUID-VkImageCreateInfo-flags-01533

If `flags` contains `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` `format` **must** be a depth or depth/stencil format

- VUID-VkImageCreateInfo-pNext-08951

If the `pNext` chain includes a `VkExternalMemoryImageCreateInfo` structure whose `handleTypes` member includes `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`, `imageType` **must** be `VK_IMAGE_TYPE_2D`

- VUID-VkImageCreateInfo-pNext-08952

If the `pNext` chain includes a `VkExternalMemoryImageCreateInfo` structure whose `handleTypes` member includes `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`, `mipLevels` **must** either be 1 or equal to the number of levels in the complete mipmap chain based on `extent.width`, `extent.height`, and `extent.depth`

- VUID-VkImageCreateInfo-pNext-08953

If the `pNext` chain includes a `VkExternalFormatQNX` structure whose `externalFormat` member is not 0, `flags` **must** not include `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`

- VUID-VkImageCreateInfo-pNext-08954

If the `pNext` chain includes a `VkExternalFormatQNX` structure whose `externalFormat` member is not 0, `usage` **must** not include any usages except `VK_IMAGE_USAGE_SAMPLED_BIT`

- VUID-VkImageCreateInfo-pNext-08955

If the `pNext` chain includes a `VkExternalFormatQNX` structure whose `externalFormat` member is not 0, `tiling` **must** be `VK_IMAGE_TILING_OPTIMAL`

- VUID-VkImageCreateInfo-format-02795

If `format` is a depth-stencil format, `usage` includes `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, and the `pNext` chain includes a `VkImageStencilUsageCreateInfo` structure, then its `VkImageStencilUsageCreateInfo::stencilUsage` member **must** also include `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- VUID-VkImageCreateInfo-format-02796

If `format` is a depth-stencil format, `usage` does not include

VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, and the pNext chain includes a VkImageStencilUsageCreateInfo structure, then its VkImageStencilUsageCreateInfo::stencilUsage member **must** also not include VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT

- VUID-VkImageCreateInfo-format-02797

If format is a depth-stencil format, usage includes VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT, and the pNext chain includes a VkImageStencilUsageCreateInfo structure, then its VkImageStencilUsageCreateInfo::stencilUsage member **must** also include VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT

- VUID-VkImageCreateInfo-format-02798

If format is a depth-stencil format, usage does not include VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT, and the pNext chain includes a VkImageStencilUsageCreateInfo structure, then its VkImageStencilUsageCreateInfo::stencilUsage member **must** also not include VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT

- VUID-VkImageCreateInfo-Format-02536

If Format is a depth-stencil format and the pNext chain includes a VkImageStencilUsageCreateInfo structure with its stencilUsage member including VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT, extent.width **must** be less than or equal to VkPhysicalDeviceLimits::maxFramebufferWidth

- VUID-VkImageCreateInfo-format-02537

If format is a depth-stencil format and the pNext chain includes a VkImageStencilUsageCreateInfo structure with its stencilUsage member including VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT, extent.height **must** be less than or equal to VkPhysicalDeviceLimits::maxFramebufferHeight

- VUID-VkImageCreateInfo-format-02538

If the shaderStorageImageMultisample feature is not enabled, format is a depth-stencil format and the pNext chain includes a VkImageStencilUsageCreateInfo structure with its stencilUsage including VK_IMAGE_USAGE_STORAGE_BIT, samples **must** be VK_SAMPLE_COUNT_1_BIT

- VUID-VkImageCreateInfo-imageType-02082

If usage includes VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR, imageType **must** be VK_IMAGE_TYPE_2D

- VUID-VkImageCreateInfo-samples-02083

If usage includes VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR, samples **must** be VK_SAMPLE_COUNT_1_BIT

- VUID-VkImageCreateInfo-pNext-06722

If a VkImageFormatListCreateInfo structure was included in the pNext chain and format is not a multi-planar format and VkImageFormatListCreateInfo::viewFormatCount is not zero, then each format in VkImageFormatListCreateInfo::pViewFormats **must** either be compatible with the format as described in the compatibility table or, if flags contains VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT, be an uncompressed format that is size-compatible with format

- VUID-VkImageCreateInfo-pNext-10062

If a VkImageFormatListCreateInfo structure was included in the pNext chain and format is

a `multi-planar format` and `flags` contains `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` and `VkImageFormatListCreateInfo::viewFormatCount` is not zero, then each format in `VkImageFormatListCreateInfo::pViewFormats` **must** be compatible with the `VkFormat` for the plane of the image format

- VUID-VkImageCreateInfo-flags-04738

If `flags` does not contain `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` and the `pNext` chain includes a `VkImageFormatListCreateInfo` structure, then `VkImageFormatListCreateInfo::viewFormatCount` **must** be 0 or 1

Valid Usage (Implicit)

- VUID-VkImageCreateInfo-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO`

- VUID-VkImageCreateInfo-pNext-pNext

Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkExternalFormatQNX`, `VkExternalMemoryImageCreateInfo`, `VkImageDrmFormatModifierExplicitCreateInfoEXT`, `VkImageDrmFormatModifierListCreateInfoEXT`, `VkImageFormatListCreateInfo`, `VkImageStencilUsageCreateInfo`, or `VkImageSwapchainCreateInfoKHR`

- VUID-VkImageCreateInfo-sType-unique

The `sType` value of each structure in the `pNext` chain **must** be unique

- VUID-VkImageCreateInfo-flags-parameter

`flags` **must** be a valid combination of `VkImageCreateFlagBits` values

- VUID-VkImageCreateInfo-imageType-parameter

`imageType` **must** be a valid `VkImageType` value

- VUID-VkImageCreateInfo-format-parameter

`format` **must** be a valid `VkFormat` value

- VUID-VkImageCreateInfo-samples-parameter

`samples` **must** be a valid `VkSampleCountFlagBits` value

- VUID-VkImageCreateInfo-tiling-parameter

`tiling` **must** be a valid `VkImageTiling` value

- VUID-VkImageCreateInfo-usage-parameter

`usage` **must** be a valid combination of `VkImageUsageFlagBits` values

- VUID-VkImageCreateInfo-usage-requiredbitmask

`usage` **must** not be 0

- VUID-VkImageCreateInfo-sharingMode-parameter

`sharingMode` **must** be a valid `VkSharingMode` value

- VUID-VkImageCreateInfo-initialLayout-parameter

`initialLayout` **must** be a valid `VkImageLayout` value

The `VkImageStencilUsageCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkImageStencilUsageCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkImageUsageFlags  stencilUsage;
} VkImageStencilUsageCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `stencilUsage` is a bitmask of `VkImageUsageFlagBits` describing the intended usage of the stencil aspect of the image.

If the `pNext` chain of `VkImageCreateInfo` includes a `VkImageStencilUsageCreateInfo` structure, then that structure includes the usage flags specific to the stencil aspect of the image for an image with a depth-stencil format.

This structure specifies image usages which only apply to the stencil aspect of a depth/stencil format image. When this structure is included in the `pNext` chain of `VkImageCreateInfo`, the stencil aspect of the image **must** only be used as specified by `stencilUsage`. When this structure is not included in the `pNext` chain of `VkImageCreateInfo`, the stencil aspect of an image **must** only be used as specified by `VkImageCreateInfo::usage`. Use of other aspects of an image are unaffected by this structure.

This structure **can** also be included in the `pNext` chain of `VkPhysicalDeviceImageFormatInfo2` to query additional capabilities specific to image creation parameter combinations including a separate set of usage flags for the stencil aspect of the image using `vkGetPhysicalDeviceImageFormatProperties2`. When this structure is not included in the `pNext` chain of `VkPhysicalDeviceImageFormatInfo2` then the implicit value of `stencilUsage` matches that of `VkPhysicalDeviceImageFormatInfo2::usage`.

Valid Usage

- VUID-VkImageStencilUsageCreateInfo-stencilUsage-02539
If `stencilUsage` includes `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`, it **must** not include bits other than `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT` or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`

Valid Usage (Implicit)

- VUID-VkImageStencilUsageCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_STENCIL_USAGE_CREATE_INFO`
- VUID-VkImageStencilUsageCreateInfo-stencilUsage-parameter
`stencilUsage` **must** be a valid combination of `VkImageUsageFlagBits` values
- VUID-VkImageStencilUsageCreateInfo-stencilUsage-requiredbitmask
`stencilUsage` **must** not be `0`

To define a set of external memory handle types that **may** be used as backing store for an image, add a `VkExternalMemoryImageCreateInfo` structure to the `pNext` chain of the `VkImageCreateInfo` structure. The `VkExternalMemoryImageCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalMemoryImageCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkExternalMemoryHandleTypeFlags handleTypes;
} VkExternalMemoryImageCreateInfo;
```

NOTE

A `VkExternalMemoryImageCreateInfo` structure with a non-zero `handleTypes` field must be included in the creation parameters for an image that will be bound to memory that is either exported or imported.

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleTypes` is zero or a bitmask of `VkExternalMemoryHandleTypeFlagBits` specifying one or more external memory handle types.

Valid Usage (Implicit)

- VUID-VkExternalMemoryImageCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO`
- VUID-VkExternalMemoryImageCreateInfo-handleTypes-parameter
`handleTypes` **must** be a valid combination of `VkExternalMemoryHandleTypeFlagBits` values

To create an image with an `QNX Screen external format`, add a `VkExternalFormatQNX` structure in the `pNext` chain of `VkImageCreateInfo`. `VkExternalFormatQNX` is defined as:

```
// Provided by VK_QNX_external_memory_screen_buffer
typedef struct VkExternalFormatQNX {
    VkStructureType    sType;
    void*              pNext;
    uint64_t           externalFormat;
} VkExternalFormatQNX;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `externalFormat` is an implementation-defined identifier for the external format

If `externalFormat` is zero, the effect is as if the `VkExternalFormatQNX` structure was not present. Otherwise, the `image` will have the specified external format.

Valid Usage

- VUID-VkExternalFormatQNX-externalFormat-08956
`externalFormat` **must** be `0` or a value returned in the `externalFormat` member of `VkScreenBufferFormatPropertiesQNX` by an earlier call to `vkGetScreenBufferPropertiesQNX`

Valid Usage (Implicit)

- VUID-VkExternalFormatQNX-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_FORMAT_QNX`

If the `pNext` chain of `VkImageCreateInfo` includes a `VkImageSwapchainCreateInfoKHR` structure, then that structure includes a swapchain handle indicating that the image will be bound to memory from that swapchain.

The `VkImageSwapchainCreateInfoKHR` structure is defined as:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
typedef struct VkImageSwapchainCreateInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkSwapchainKHR     swapchain;
} VkImageSwapchainCreateInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `swapchain` is `VK_NULL_HANDLE` or a handle of a swapchain that the image will be bound to.

Valid Usage

- VUID-VkImageSwapchainCreateInfoKHR-swapchain-00995
If `swapchain` is not `VK_NULL_HANDLE`, the fields of `VkImageCreateInfo` **must** match the `implied image creation parameters` of the swapchain

Valid Usage (Implicit)

- VUID-VkImageSwapchainCreateInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_SWAPCHAIN_CREATE_INFO_KHR`
- VUID-VkImageSwapchainCreateInfoKHR-swapchain-parameter
If `swapchain` is not `VK_NULL_HANDLE`, `swapchain` **must** be a valid `VkSwapchainKHR` handle

If the `pNext` chain of `VkImageCreateInfo` includes a `VkImageFormatListCreateInfo` structure, then that structure contains a list of all formats that **can** be used when creating views of this image.

The `VkImageFormatListCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkImageFormatListCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           viewFormatCount;
    const VkFormat*     pViewFormats;
} VkImageFormatListCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `viewFormatCount` is the number of entries in the `pViewFormats` array.
- `pViewFormats` is a pointer to an array of `VkFormat` values specifying all formats which **can** be used when creating views of this image.

If `viewFormatCount` is zero, `pViewFormats` is ignored and the image is created as if the `VkImageFormatListCreateInfo` structure were not included in the `pNext` chain of `VkImageCreateInfo`.

Valid Usage

- VUID-VkImageFormatListCreateInfo-viewFormatCount-09540
If `viewFormatCount` is not 0, each element of `pViewFormats` **must** not be `VK_FORMAT_UNDEFINED`

Valid Usage (Implicit)

- VUID-VkImageFormatListCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO`
- VUID-VkImageFormatListCreateInfo-pViewFormats-parameter
If `viewFormatCount` is not 0, `pViewFormats` **must** be a valid pointer to an array of `viewFormatCount` valid `VkFormat` values

If the `pNext` chain of `VkImageCreateInfo` includes a `VkImageDrmFormatModifierListCreateInfoEXT` structure, then the image will be created with one of the [Linux DRM format modifiers](#) listed in the structure. The choice of modifier is implementation-dependent.

The `VkImageDrmFormatModifierListCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier
typedef struct VkImageDrmFormatModifierListCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           drmFormatModifierCount;
    const uint64_t*     pDrmFormatModifiers;
} VkImageDrmFormatModifierListCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `drmFormatModifierCount` is the length of the `pDrmFormatModifiers` array.
- `pDrmFormatModifiers` is a pointer to an array of *Linux DRM format modifiers*.

Valid Usage

- VUID-VkImageDrmFormatModifierListCreateInfoEXT-pDrmFormatModifiers-02263
Each *modifier* in `pDrmFormatModifiers` **must** be compatible with the parameters in `VkImageCreateInfo` and its `pNext` chain, as determined by querying `VkPhysicalDeviceImageFormatInfo2` extended with `VkPhysicalDeviceImageDrmFormatModifierInfoEXT`

Valid Usage (Implicit)

- VUID-VkImageDrmFormatModifierListCreateInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_LIST_CREATE_INFO_EXT`
- VUID-VkImageDrmFormatModifierListCreateInfoEXT-pDrmFormatModifiers-parameter
`pDrmFormatModifiers` **must** be a valid pointer to an array of `drmFormatModifierCount` `uint64_t` values
- VUID-VkImageDrmFormatModifierListCreateInfoEXT-drmFormatModifierCount-arraylength
`drmFormatModifierCount` **must** be greater than 0

If the `pNext` chain of `VkImageCreateInfo` includes a `VkImageDrmFormatModifierExplicitCreateInfoEXT` structure, then the image will be created with the *Linux DRM format modifier* and memory layout defined by the structure.

The `VkImageDrmFormatModifierExplicitCreateInfoEXT` structure is defined as:


```
// Provided by VK_EXT_image_drm_format_modifier
typedef struct VkImageDrmFormatModifierExplicitCreateInfoEXT {
    VkStructureType      sType;
    const void*          pNext;
    uint64_t             drmFormatModifier;
    uint32_t             drmFormatModifierPlaneCount;
    const VkSubresourceLayout* pPlaneLayouts;
} VkImageDrmFormatModifierExplicitCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `drmFormatModifier` is the *Linux DRM format modifier* with which the image will be created.
- `drmFormatModifierPlaneCount` is the number of *memory planes* in the image (as reported by `VkDrmFormatModifierPropertiesEXT`) as well as the length of the `pPlaneLayouts` array.
- `pPlaneLayouts` is a pointer to an array of `VkSubresourceLayout` structures describing the image's *memory planes*.

The i^{th} member of `pPlaneLayouts` describes the layout of the image's i^{th} *memory plane* (that is, `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT`). In each element of `pPlaneLayouts`, the implementation **must** ignore `size`. The implementation calculates the size of each plane, which the application **can** query with `vkGetImageSubresourceLayout`.

When creating an image with `VkImageDrmFormatModifierExplicitCreateInfoEXT`, it is the application's responsibility to satisfy all valid usage requirements. However, the implementation **must** validate that the provided `pPlaneLayouts`, when combined with the provided `drmFormatModifier` and other creation parameters in `VkImageCreateInfo` and its `pNext` chain, produce a valid image. (This validation is necessarily implementation-dependent and outside the scope of Vulkan, and therefore not described by valid usage requirements). If this validation fails, then `vkCreateImage` returns `VK_ERROR_INVALID_DRM_FORMAT_MODIFIER_PLANE_LAYOUT_EXT`.

Valid Usage

- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-drmFormatModifier-02264
`drmFormatModifier` **must** be compatible with the parameters in `VkImageCreateInfo` and its `pNext` chain, as determined by querying `VkPhysicalDeviceImageFormatInfo2` extended with `VkPhysicalDeviceImageDrmFormatModifierInfoEXT`
- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-drmFormatModifierPlaneCount-02265
`drmFormatModifierPlaneCount` **must** be equal to the `VkDrmFormatModifierPropertiesEXT::drmFormatModifierPlaneCount` associated with `VkImageCreateInfo::format` and `drmFormatModifier`, as found by querying `VkDrmFormatModifierPropertiesListEXT`
- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-size-02267
For each element of `pPlaneLayouts`, `size` **must** be 0
- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-arrayPitch-02268

For each element of `pPlaneLayouts`, `arrayPitch` **must** be 0 if `VkImageCreateInfo::arrayLayers` is 1

- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-depthPitch-02269
For each element of `pPlaneLayouts`, `depthPitch` **must** be 0 if `VkImageCreateInfo::extent.depth` is 1

Valid Usage (Implicit)

- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_EXPLICIT_CREATE_INFO_EXT`
- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-pPlaneLayouts-parameter
`pPlaneLayouts` **must** be a valid pointer to an array of `drmFormatModifierPlaneCount` `VkSubresourceLayout` structures
- VUID-VkImageDrmFormatModifierExplicitCreateInfoEXT-drmFormatModifierPlaneCount-arraylength
`drmFormatModifierPlaneCount` **must** be greater than 0

Bits which **can** be set in

- `VkImageViewUsageCreateInfo::usage`
- `VkImageStencilUsageCreateInfo::stencilUsage`
- `VkImageCreateInfo::usage`

specify intended usage of an image, and are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageUsageFlagBits {
    VK_IMAGE_USAGE_TRANSFER_SRC_BIT = 0x00000001,
    VK_IMAGE_USAGE_TRANSFER_DST_BIT = 0x00000002,
    VK_IMAGE_USAGE_SAMPLED_BIT = 0x00000004,
    VK_IMAGE_USAGE_STORAGE_BIT = 0x00000008,
    VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT = 0x00000010,
    VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT = 0x00000020,
    VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT = 0x00000040,
    VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT = 0x00000080,
    // Provided by VK_KHR_fragment_shading_rate
    VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR = 0x00000100,
} VkImageUsageFlagBits;
```

- `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` specifies that the image **can** be used as the source of a transfer command.
- `VK_IMAGE_USAGE_TRANSFER_DST_BIT` specifies that the image **can** be used as the destination of a transfer command.
- `VK_IMAGE_USAGE_SAMPLED_BIT` specifies that the image **can** be used to create a `VkImageView` suitable

for occupying a `VkDescriptorSet` slot either of type `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE` or `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, and be sampled by a shader.

- `VK_IMAGE_USAGE_STORAGE_BIT` specifies that the image **can** be used to create a `VkImageView` suitable for occupying a `VkDescriptorSet` slot of type `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`.
- `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` specifies that the image **can** be used to create a `VkImageView` suitable for use as a color or resolve attachment in a `VkFramebuffer`.
- `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT` specifies that the image **can** be used to create a `VkImageView` suitable for use as a depth/stencil or depth/stencil resolve attachment in a `VkFramebuffer`.
- `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT` specifies that implementations **may** support using [memory allocations](#) with the `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT` to back an image with this usage. This bit **can** be set for any image that **can** be used to create a `VkImageView` suitable for use as a color, resolve, depth/stencil, or input attachment.
- `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT` specifies that the image **can** be used to create a `VkImageView` suitable for occupying `VkDescriptorSet` slot of type `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`; be read from a shader as an input attachment; and be used as an input attachment in a framebuffer.
- `VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR` specifies that the image **can** be used to create a `VkImageView` suitable for use as a [fragment shading rate attachment](#)

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkImageUsageFlags;
```

`VkImageUsageFlags` is a bitmask type for setting a mask of zero or more [VkImageUsageFlagBits](#).

When creating a `VkImageView` one of the following [VkImageUsageFlagBits](#) **must** be set:

- `VK_IMAGE_USAGE_SAMPLED_BIT`
- `VK_IMAGE_USAGE_STORAGE_BIT`
- `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
- `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`
- `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`
- `VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`

Bits which **can** be set in `VkImageCreateInfo::flags`, specifying additional parameters of an image, are:

```

// Provided by VK_VERSION_1_0
typedef enum VkImageCreateFlagBits {
    VK_IMAGE_CREATE_SPARSE_BINDING_BIT = 0x00000001,
    VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT = 0x00000002,
    VK_IMAGE_CREATE_SPARSE_ALIASED_BIT = 0x00000004,
    VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT = 0x00000008,
    VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT = 0x00000010,
// Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_ALIAS_BIT = 0x00000400,
// Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT = 0x00000040,
// Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT = 0x00000020,
// Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT = 0x00000080,
// Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_EXTENDED_USAGE_BIT = 0x00000100,
// Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_PROTECTED_BIT = 0x00000800,
// Provided by VK_VERSION_1_1
    VK_IMAGE_CREATE_DISJOINT_BIT = 0x00000200,
// Provided by VK_EXT_sample_locations
    VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT = 0x00001000,
} VkImageCreateFlagBits;

```

- **VK_IMAGE_CREATE_SPARSE_BINDING_BIT** specifies that the image will be backed using sparse memory binding. This flag is not supported in Vulkan SC [\[SCID-8\]](#).
- **VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT** specifies that the image **can** be partially backed using sparse memory binding. Images created with this flag **must** also be created with the **VK_IMAGE_CREATE_SPARSE_BINDING_BIT** flag. This flag is not supported in Vulkan SC [\[SCID-8\]](#).
- **VK_IMAGE_CREATE_SPARSE_ALIASED_BIT** specifies that the image will be backed using sparse memory binding with memory ranges that might also simultaneously be backing another image (or another portion of the same image). Images created with this flag **must** also be created with the **VK_IMAGE_CREATE_SPARSE_BINDING_BIT** flag. This flag is not supported in Vulkan SC [\[SCID-8\]](#).
- **VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT** specifies that the image **can** be used to create a **VkImageView** with a different format from the image. For **multi-planar formats**, **VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT** specifies that a **VkImageView** can be created of a *plane* of the image.
- **VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT** specifies that the image **can** be used to create a **VkImageView** of type **VK_IMAGE_VIEW_TYPE_CUBE** or **VK_IMAGE_VIEW_TYPE_CUBE_ARRAY**.
- **VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT** specifies that the image **can** be used to create a **VkImageView** of type **VK_IMAGE_VIEW_TYPE_2D** or **VK_IMAGE_VIEW_TYPE_2D_ARRAY**.
- **VK_IMAGE_CREATE_PROTECTED_BIT** specifies that the image is a protected image.
- **VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT** specifies that the image **can** be used with a non-zero value of the **splitInstanceBindRegionCount** member of a

[VkBindImageMemoryDeviceGroupInfo](#) structure passed into [vkBindImageMemory2](#). This flag also has the effect of making the image use the standard sparse image block dimensions. This flag is not supported in Vulkan SC [\[SCID-8\]](#).

- [VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT](#) specifies that the image having a compressed format **can** be used to create a [VkImageView](#) with an uncompressed format where each texel in the image view corresponds to a compressed texel block of the image.
- [VK_IMAGE_CREATE_EXTENDED_USAGE_BIT](#) specifies that the image **can** be created with usage flags that are not supported for the format the image is created with but are supported for at least one format a [VkImageView](#) created from the image **can** have.
- [VK_IMAGE_CREATE_DISJOINT_BIT](#) specifies that an image with a [multi-planar format](#) **must** have each plane separately bound to memory, rather than having a single memory binding for the whole image; the presence of this bit distinguishes a *disjoint image* from an image without this bit set.
- [VK_IMAGE_CREATE_ALIAS_BIT](#) specifies that two images created with the same creation parameters and aliased to the same memory **can** interpret the contents of the memory consistently with each other, subject to the rules described in the [Memory Aliasing](#) section. This flag further specifies that each plane of a *disjoint* image **can** share an in-memory non-linear representation with single-plane images, and that a single-plane image **can** share an in-memory non-linear representation with a plane of a multi-planar disjoint image, according to the rules in [Compatible Formats of Planes of Multi-Planar Formats](#). If the [pNext](#) chain includes a [VkExternalMemoryImageCreateInfo](#) structure whose [handleTypes](#) member is not `0`, it is as if [VK_IMAGE_CREATE_ALIAS_BIT](#) is set.
- [VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT](#) specifies that an image with a depth or depth/stencil format **can** be used with custom sample locations when used as a depth/stencil attachment.

See [Sparse Resource Features](#) and [Sparse Physical Device Features](#) for more details.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkImageCreateFlags;
```

[VkImageCreateFlags](#) is a bitmask type for setting a mask of zero or more [VkImageCreateFlagBits](#).

Possible values of [VkImageCreateInfo::imageType](#), specifying the basic dimensionality of an image, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageType {
    VK_IMAGE_TYPE_1D = 0,
    VK_IMAGE_TYPE_2D = 1,
    VK_IMAGE_TYPE_3D = 2,
} VkImageType;
```

- [VK_IMAGE_TYPE_1D](#) specifies a one-dimensional image.

- `VK_IMAGE_TYPE_2D` specifies a two-dimensional image.
- `VK_IMAGE_TYPE_3D` specifies a three-dimensional image.

Possible values of `VkImageCreateInfo::tiling`, specifying the tiling arrangement of texel blocks in an image, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageTiling {
    VK_IMAGE_TILING_OPTIMAL = 0,
    VK_IMAGE_TILING_LINEAR = 1,
    // Provided by VK_EXT_image_drm_format_modifier
    VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT = 1000158000,
} VkImageTiling;
```

- `VK_IMAGE_TILING_OPTIMAL` specifies optimal tiling (texels are laid out in an implementation-dependent arrangement, for more efficient memory access).
- `VK_IMAGE_TILING_LINEAR` specifies linear tiling (texels are laid out in memory in row-major order, possibly with some padding on each row).
- `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` specifies that the image's tiling is defined by a [Linux DRM format modifier](#). The modifier is specified at image creation with `VkImageDrmFormatModifierListCreateInfoEXT` or `VkImageDrmFormatModifierExplicitCreateInfoEXT`, and **can** be queried with `vkGetImageDrmFormatModifierPropertiesEXT`.

To query the memory layout of an image subresource, call:

```
// Provided by VK_VERSION_1_0
void vkGetImageSubresourceLayout(
    VkDevice          device,
    VkImage           image,
    const VkImageSubresource* pSubresource,
    VkSubresourceLayout* pLayout);
```

- `device` is the logical device that owns the image.
- `image` is the image whose layout is being queried.
- `pSubresource` is a pointer to a `VkImageSubresource` structure selecting a specific image subresource from the image.
- `pLayout` is a pointer to a `VkSubresourceLayout` structure in which the layout is returned.

If the image is [linear](#), then the returned layout is valid for [host access](#).

If the image's tiling is `VK_IMAGE_TILING_LINEAR` and its format is a [multi-planar format](#), then `vkGetImageSubresourceLayout` describes one *format plane* of the image. If the image's tiling is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `vkGetImageSubresourceLayout` describes one *memory plane* of the image. If the image's tiling is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` and the image is

`non-linear`, then the returned layout has an implementation-dependent meaning; the vendor of the image's `DRM format modifier` **may** provide documentation that explains how to interpret the returned layout.

`vkGetImageSubresourceLayout` is invariant for the lifetime of a single image. However, the subresource layout of images in Android hardware buffer or QNX Screen buffer external memory is not known until the image has been bound to memory, so applications **must** not call `vkGetImageSubresourceLayout` for such an image before it has been bound.

Valid Usage

- VUID-vkGetImageSubresourceLayout-image-07790
The `image` **must** have been created with `tiling` equal to `VK_IMAGE_TILING_LINEAR` or `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`
- VUID-vkGetImageSubresourceLayout-aspectMask-00997
The `aspectMask` member of `pSubresource` **must** only have a single bit set
- VUID-vkGetImageSubresourceLayout-mipLevel-01716
The `mipLevel` member of `pSubresource` **must** be less than the `mipLevels` specified in `image`
- VUID-vkGetImageSubresourceLayout-arrayLayer-01717
The `arrayLayer` member of `pSubresource` **must** be less than the `arrayLayers` specified in `image`
- VUID-vkGetImageSubresourceLayout-format-08886
If `format` of the `image` is a color format that is not a `multi-planar format`, and `tiling` of the `image` is `VK_IMAGE_TILING_LINEAR` or `VK_IMAGE_TILING_OPTIMAL`, the `aspectMask` member of `pSubresource` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-vkGetImageSubresourceLayout-format-04462
If `format` of the `image` has a depth component, the `aspectMask` member of `pSubresource` **must** contain `VK_IMAGE_ASPECT_DEPTH_BIT`
- VUID-vkGetImageSubresourceLayout-format-04463
If `format` of the `image` has a stencil component, the `aspectMask` member of `pSubresource` **must** contain `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-vkGetImageSubresourceLayout-format-04464
If `format` of the `image` does not contain a stencil or depth component, the `aspectMask` member of `pSubresource` **must** not contain `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-vkGetImageSubresourceLayout-tiling-08717
If the `tiling` of the `image` is `VK_IMAGE_TILING_LINEAR` and has a `multi-planar format`, then the `aspectMask` member of `pSubresource` **must** be a single valid `multi-planar aspect mask` bit
- VUID-vkGetImageSubresourceLayout-tiling-09433
If the `tiling` of the `image` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then the `aspectMask` member of `pSubresource` **must** be `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT` and the index `i` **must** be less than the `VkDrmFormatModifierPropertiesEXT::drmFormatModifierPlaneCount` associated with the image's `format` and `VkImageDrmFormatModifierPropertiesEXT::drmFormatModifier`

Valid Usage (Implicit)

- VUID-vkGetImageSubresourceLayout-device-parameter
device must be a valid [VkDevice](#) handle
- VUID-vkGetImageSubresourceLayout-image-parameter
image must be a valid [VkImage](#) handle
- VUID-vkGetImageSubresourceLayout-pSubresource-parameter
pSubresource must be a valid pointer to a valid [VkImageSubresource](#) structure
- VUID-vkGetImageSubresourceLayout-pLayout-parameter
pLayout must be a valid pointer to a [VkSubresourceLayout](#) structure
- VUID-vkGetImageSubresourceLayout-image-parent
image must have been created, allocated, or retrieved from **device**

The [VkImageSubresource](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageSubresource {
    VkImageAspectFlags    aspectMask;
    uint32_t              mipLevel;
    uint32_t              arrayLayer;
} VkImageSubresource;
```

- **aspectMask** is a [VkImageAspectFlags](#) value selecting the image *aspect*.
- **mipLevel** selects the mipmap level.
- **arrayLayer** selects the array layer.

Valid Usage (Implicit)

- VUID-VkImageSubresource-aspectMask-parameter
aspectMask must be a valid combination of [VkImageAspectFlagBits](#) values
- VUID-VkImageSubresource-aspectMask-requiredbitmask
aspectMask must not be 0

Information about the layout of the image subresource is returned in a [VkSubresourceLayout](#) structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkSubresourceLayout {
    VkDeviceSize    offset;
    VkDeviceSize    size;
    VkDeviceSize    rowPitch;
    VkDeviceSize    arrayPitch;
    VkDeviceSize    depthPitch;
} VkSubresourceLayout;
```

- **offset** is the byte offset from the start of the image or the plane where the image subresource begins.
- **size** is the size in bytes of the image subresource. **size** includes any extra memory that is required based on **rowPitch**.
- **rowPitch** describes the number of bytes between each row of texels in an image.
- **arrayPitch** describes the number of bytes between each array layer of an image.
- **depthPitch** describes the number of bytes between each slice of 3D image.

If the image is **linear**, then **rowPitch**, **arrayPitch** and **depthPitch** describe the layout of the image subresource in linear memory. For uncompressed formats, **rowPitch** is the number of bytes between texels with the same x coordinate in adjacent rows (y coordinates differ by one). **arrayPitch** is the number of bytes between texels with the same x and y coordinate in adjacent array layers of the image (array layer values differ by one). **depthPitch** is the number of bytes between texels with the same x and y coordinate in adjacent slices of a 3D image (z coordinates differ by one). Expressed as an addressing formula, the starting byte of a texel in the image subresource has address:

```
// (x,y,z,layer) are in texel coordinates
address(x,y,z,layer) = layer*arrayPitch + z*depthPitch + y*rowPitch + x*elementSize +
offset
```

For compressed formats, the **rowPitch** is the number of bytes between compressed texel blocks in adjacent rows. **arrayPitch** is the number of bytes between compressed texel blocks in adjacent array layers. **depthPitch** is the number of bytes between compressed texel blocks in adjacent slices of a 3D image.

```
// (x,y,z,layer) are in compressed texel block coordinates
address(x,y,z,layer) = layer*arrayPitch + z*depthPitch + y*rowPitch + x
*compressedTexelBlockByteSize + offset;
```

The value of **arrayPitch** is undefined for images that were not created as arrays. **depthPitch** is defined only for 3D images.

If the image has a *single-plane* color format and its tiling is **VK_IMAGE_TILING_LINEAR**, then the **aspectMask** member of **VkImageSubresource** **must** be **VK_IMAGE_ASPECT_COLOR_BIT**.

If the image has a depth/stencil format and its tiling is **VK_IMAGE_TILING_LINEAR**, then **aspectMask**

must be either `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`. On implementations that store depth and stencil aspects separately, querying each of these image subresource layouts will return a different **offset** and **size** representing the region of memory used for that aspect. On implementations that store depth and stencil aspects interleaved, the same **offset** and **size** are returned and represent the interleaved memory allocation.

If the image has a **multi-planar format** and its tiling is `VK_IMAGE_TILING_LINEAR`, then the **aspectMask** member of `VkImageSubresource` **must** be `VK_IMAGE_ASPECT_PLANE_0_BIT`, `VK_IMAGE_ASPECT_PLANE_1_BIT`, or (for 3-plane formats only) `VK_IMAGE_ASPECT_PLANE_2_BIT`. Querying each of these image subresource layouts will return a different **offset** and **size** representing the region of memory used for that plane. If the image is *disjoint*, then the **offset** is relative to the base address of the plane. If the image is *non-disjoint*, then the **offset** is relative to the base address of the image.

If the image's tiling is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then the **aspectMask** member of `VkImageSubresource` **must** be one of `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT`, where the maximum allowed plane index *i* is defined by the `VkDrmFormatModifierPropertiesEXT::drmFormatModifierPlaneCount` associated with the image's `VkImageCreateInfo::format` and **modifier**. The memory range used by the subresource is described by **offset** and **size**. If the image is *disjoint*, then the **offset** is relative to the base address of the *memory plane*. If the image is *non-disjoint*, then the **offset** is relative to the base address of the image. If the image is **non-linear**, then **rowPitch**, **arrayPitch**, and **depthPitch** have an implementation-dependent meaning.

If an image was created with `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then the image has a **Linux DRM format modifier**. To query the *modifier*, call:

```
// Provided by VK_EXT_image_drm_format_modifier
VkResult vkGetImageDrmFormatModifierPropertiesEXT(
    VkDevice          device,
    VkImage           image,
    VkImageDrmFormatModifierPropertiesEXT* pProperties);
```

- **device** is the logical device that owns the image.
- **image** is the queried image.
- **pProperties** is a pointer to a `VkImageDrmFormatModifierPropertiesEXT` structure in which properties of the image's *DRM format modifier* are returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetImageDrmFormatModifierPropertiesEXT` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetImageDrmFormatModifierPropertiesEXT-image-02272
image **must** have been created with **tiling** equal to `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`

Valid Usage (Implicit)

- VUID-vkGetImageDrmFormatModifierPropertiesEXT-device-parameter **device** **must** be a valid [VkDevice](#) handle
- VUID-vkGetImageDrmFormatModifierPropertiesEXT-image-parameter **image** **must** be a valid [VkImage](#) handle
- VUID-vkGetImageDrmFormatModifierPropertiesEXT-pProperties-parameter **pProperties** **must** be a valid pointer to a [VkImageDrmFormatModifierPropertiesEXT](#) structure
- VUID-vkGetImageDrmFormatModifierPropertiesEXT-image-parent **image** **must** have been created, allocated, or retrieved from **device**

Return Codes

Success

- [VK_SUCCESS](#)

Failure

- [VK_ERROR_OUT_OF_HOST_MEMORY](#)
- [VK_ERROR_UNKNOWN](#)
- [VK_ERROR_VALIDATION_FAILED](#)

The [VkImageDrmFormatModifierPropertiesEXT](#) structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier
typedef struct VkImageDrmFormatModifierPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint64_t           drmFormatModifier;
} VkImageDrmFormatModifierPropertiesEXT;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is [NULL](#) or a pointer to a structure extending this structure.
- **drmFormatModifier** returns the image's [Linux DRM format modifier](#).

If the **image** was created with [VkImageDrmFormatModifierListCreateInfoEXT](#), then the returned **drmFormatModifier** **must** belong to the list of modifiers provided at time of image creation in [VkImageDrmFormatModifierListCreateInfoEXT::pDrmFormatModifiers](#). If the **image** was created with [VkImageDrmFormatModifierExplicitCreateInfoEXT](#), then the returned **drmFormatModifier** **must** be the modifier provided at time of image creation in [VkImageDrmFormatModifierExplicitCreateInfoEXT::drmFormatModifier](#).

Valid Usage (Implicit)

- VUID-VkImageDrmFormatModifierPropertiesEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_PROPERTIES_EXT`
- VUID-VkImageDrmFormatModifierPropertiesEXT-pNext-pNext
`pNext` **must** be `NULL`

To destroy an image, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyImage(
    VkDevice          device,
    VkImage            image,
    const VkAllocationCallbacks* pAllocator);
```

- `device` is the logical device that destroys the image.
- `image` is the image to destroy.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage

- VUID-vkDestroyImage-image-01000
All submitted commands that refer to `image`, either directly or via a `VkImageView`, **must** have completed execution
- VUID-vkDestroyImage-image-04882
`image` **must** not have been acquired from `vkGetSwapchainImagesKHR`

Valid Usage (Implicit)

- VUID-vkDestroyImage-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkDestroyImage-image-parameter
If `image` is not `VK_NULL_HANDLE`, `image` **must** be a valid `VkImage` handle
- VUID-vkDestroyImage-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkDestroyImage-image-parent
If `image` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `image` **must** be externally synchronized

12.4.1. Image Format Features

Valid uses of a `VkImage` may depend on the image's *format features*, defined below. Such constraints are documented in the affected valid usage statement.

- If the image was created with `VK_IMAGE_TILING_LINEAR`, then its set of *format features* is the value of `VkFormatProperties::linearTilingFeatures` found by calling `vkGetPhysicalDeviceFormatProperties` on the same `format` as `VkImageCreateInfo::format`.
- If the image was created with `VK_IMAGE_TILING_OPTIMAL`, but without a `QNX Screen Buffer external format` then its set of *format features* is the value of `VkFormatProperties::optimalTilingFeatures` found by calling `vkGetPhysicalDeviceFormatProperties` on the same `format` as `VkImageCreateInfo::format`.
- If the image was created with an `QNX Screen buffer external format`, then its set of *format features* is the value of `VkScreenBufferFormatPropertiesQNX::formatFeatures` found by calling `vkGetScreenBufferPropertiesQNX` on the QNX Screen buffer that was imported to the `VkDeviceMemory` to which the image is bound.
- If the image was created with `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then:
 - The image's DRM format modifier is the value of `VkImageDrmFormatModifierPropertiesEXT::drmFormatModifier` found by calling `vkGetImageDrmFormatModifierPropertiesEXT`.
 - Let `VkDrmFormatModifierPropertiesListEXT::pDrmFormatModifierProperties` be the array found by calling `vkGetPhysicalDeviceFormatProperties2` on the same `format` as `VkImageCreateInfo::format`.
 - Let `VkDrmFormatModifierPropertiesEXT prop` be the array element whose `drmFormatModifier` member is the value of the image's DRM format modifier.
 - Then the image's set of *format features* is the value of `prop::drmFormatModifierTilingFeatures`.

12.4.2. Image Mip Level Sizing

A *complete mipmap chain* is the full set of mip levels, from the largest mip level provided, down to the *minimum mip level size*.

Conventional Images

For conventional images, the dimensions of each successive mip level, $n+1$, are:

$$\text{width}_{n+1} = \max(\lfloor \text{width}_n / 2 \rfloor, 1)$$

$$\text{height}_{n+1} = \max(\lfloor \text{height}_n / 2 \rfloor, 1)$$

$$\text{depth}_{n+1} = \max(\lfloor \text{depth}_n / 2 \rfloor, 1)$$

where width_n , height_n , and depth_n are the dimensions of the next larger mip level, n .

The minimum mip level size is:

- 1 for one-dimensional images,
- 1x1 for two-dimensional images, and
- 1x1x1 for three-dimensional images.

The number of levels in a complete mipmap chain is:

$$\lceil \log_2(\max(\text{width}_0, \text{height}_0, \text{depth}_0)) \rceil + 1$$

where width_0 , height_0 , and depth_0 are the dimensions of the largest (most detailed) mip level, 0.

12.5. Image Layouts

Images are stored in implementation-dependent opaque layouts in memory. Each layout has limitations on what kinds of operations are supported for image subresources using the layout. At any given time, the data representing an image subresource in memory exists in a particular layout which is determined by the most recent layout transition that was performed on that image subresource. Applications have control over which layout each image subresource uses, and **can** transition an image subresource from one layout to another. Transitions **can** happen with an image memory barrier, included as part of a [vkCmdPipelineBarrier](#) or a [vkCmdWaitEvents](#) command buffer command (see [Image Memory Barriers](#)), or as part of a subpass dependency within a render pass (see [VkSubpassDependency](#)).

Image layout is per-image subresource. Separate image subresources of the same image **can** be in different layouts at the same time, with the exception that depth and stencil aspects of a given image subresource **can** only be in different layouts if the [separateDepthStencilLayouts](#) feature is enabled. When an [VkImageView](#) descriptor is accessed on the device, all image subresources must be in a valid image layout.

NOTE

Each layout **may** offer optimal performance for a specific usage of image memory. For example, an image with a layout of [VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL](#) **may** provide optimal performance for use as a color attachment, but be unsupported for use in transfer commands. Applications **can** transition an image subresource from one layout to another in order to achieve optimal performance when the image subresource is used for multiple kinds of operations. After initialization, applications need not use any layout other than the general layout, though this **may** produce suboptimal performance on some implementations.

Upon creation, all image subresources of an image are initially in the same layout, where that layout is selected by the [VkImageCreateInfo::initialLayout](#) member. The [initialLayout](#) **must** be either [VK_IMAGE_LAYOUT_UNDEFINED](#) or [VK_IMAGE_LAYOUT_PREINITIALIZED](#). If it is [VK_IMAGE_LAYOUT_PREINITIALIZED](#), then the image data **can** be preinitialized by the host while using this layout, and the transition away from this layout will preserve that data. If it is [VK_IMAGE_LAYOUT_UNDEFINED](#), then the contents of the data are considered to be undefined, and the transition away from this layout is not guaranteed to preserve that data. For either of these initial

layouts, any image subresources **must** be transitioned to another layout before they are accessed by the device.

Host access to image memory is only well-defined for [linear](#) images and for image subresources of those images which are currently in any of the following layouts:

- [VK_IMAGE_LAYOUT_PREINITIALIZED](#)
- [VK_IMAGE_LAYOUT_GENERAL](#)

Calling [vkGetImageSubresourceLayout](#) for a linear image returns a subresource layout mapping that is valid for either of those image layouts.

The set of image layouts consists of:

```

// Provided by VK_VERSION_1_0
typedef enum VkImageLayout {
    VK_IMAGE_LAYOUT_UNDEFINED = 0,
    VK_IMAGE_LAYOUT_GENERAL = 1,
    VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL = 2,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL = 3,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL = 4,
    VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL = 5,
    VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL = 6,
    VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL = 7,
    VK_IMAGE_LAYOUT_PREINITIALIZED = 8,
// Provided by VK_VERSION_1_1
    VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL = 1000117000,
// Provided by VK_VERSION_1_1
    VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL = 1000117001,
// Provided by VK_VERSION_1_2
    VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL = 1000241000,
// Provided by VK_VERSION_1_2
    VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL = 1000241001,
// Provided by VK_VERSION_1_2
    VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL = 1000241002,
// Provided by VK_VERSION_1_2
    VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL = 1000241003,
    VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL = 1000314000,
    VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL = 1000314001,
// Provided by VK_KHR_swapchain
    VK_IMAGE_LAYOUT_PRESENT_SRC_KHR = 1000001002,
// Provided by VK_KHR_shared_presentable_image
    VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR = 1000111000,
// Provided by VK_KHR_fragment_shading_rate
    VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR = 1000164003,
// Provided by VK_KHR_synchronization2
    VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR = VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL,
// Provided by VK_KHR_synchronization2
    VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR = VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL,
} VkImageLayout;

```

The type(s) of device access supported by each layout are:

- **VK_IMAGE_LAYOUT_UNDEFINED** specifies that the layout is unknown. Image memory **cannot** be transitioned into this layout. This layout **can** be used as the **initialLayout** member of **VkImageCreateInfo**. This layout **can** be used in place of the current image layout in a layout transition, but doing so will cause the contents of the image's memory to be undefined.
- **VK_IMAGE_LAYOUT_PREINITIALIZED** specifies that an image's memory is in a defined layout and **can** be populated by data, but that it has not yet been initialized by the driver. Image memory **cannot** be transitioned into this layout. This layout **can** be used as the **initialLayout** member of **VkImageCreateInfo**. This layout is intended to be used as the initial layout for an image whose contents are written by the host, and hence the data **can** be written to memory immediately, without first executing a layout transition. Currently, **VK_IMAGE_LAYOUT_PREINITIALIZED** is only

useful with `linear` images because there is not a standard layout defined for `VK_IMAGE_TILING_OPTIMAL` images.

- `VK_IMAGE_LAYOUT_GENERAL` supports all types of device access, unless specified otherwise.
- `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL` specifies a layout that **must** only be used with attachment accesses in the graphics pipeline.
- `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL` specifies a layout allowing read only access as an attachment, or in shaders as a sampled image, combined image/sampler, or input attachment.
- `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` **must** only be used as a color or resolve attachment in a `VkFramebuffer`. This layout is valid only for image subresources of images created with the `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` usage bit enabled.
- `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` specifies a layout for both the depth and stencil aspects of a depth/stencil format image allowing read and write access as a depth/stencil attachment. It is equivalent to `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` and `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`.
- `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL` specifies a layout for both the depth and stencil aspects of a depth/stencil format image allowing read only access as a depth/stencil attachment or in shaders as a sampled image, combined image/sampler, or input attachment. It is equivalent to `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL` and `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`.
- `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL` specifies a layout for depth/stencil format images allowing read and write access to the stencil aspect as a stencil attachment, and read only access to the depth aspect as a depth attachment or in shaders as a sampled image, combined image/sampler, or input attachment. It is equivalent to `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL` and `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`.
- `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL` specifies a layout for depth/stencil format images allowing read and write access to the depth aspect as a depth attachment, and read only access to the stencil aspect as a stencil attachment or in shaders as a sampled image, combined image/sampler, or input attachment. It is equivalent to `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` and `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`.
- `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` specifies a layout for the depth aspect of a depth/stencil format image allowing read and write access as a depth attachment.
- `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL` specifies a layout for the depth aspect of a depth/stencil format image allowing read-only access as a depth attachment or in shaders as a sampled image, combined image/sampler, or input attachment.
- `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` specifies a layout for the stencil aspect of a depth/stencil format image allowing read and write access as a stencil attachment.
- `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL` specifies a layout for the stencil aspect of a depth/stencil format image allowing read-only access as a stencil attachment or in shaders as a sampled image, combined image/sampler, or input attachment.
- `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL` specifies a layout allowing read-only access in a shader as a sampled image, combined image/sampler, or input attachment. This layout is valid only for image subresources of images created with the `VK_IMAGE_USAGE_SAMPLED_BIT` or `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT` usage bits enabled.

- `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` **must** only be used as a source image of a transfer command (see the definition of `VK_PIPELINE_STAGE_TRANSFER_BIT`). This layout is valid only for image subresources of images created with the `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` usage bit enabled.
- `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` **must** only be used as a destination image of a transfer command. This layout is valid only for image subresources of images created with the `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage bit enabled.
- `VK_IMAGE_LAYOUT_PRESENT_SRC_KHR` **must** only be used for presenting a presentable image for display.
- `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR` is valid only for shared presentable images, and **must** be used for any usage the image supports.
- `VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR` **must** only be used as a [fragment shading rate attachment](#) or This layout is valid only for image subresources of images created with the `VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR` usage bit enabled.

The layout of each image subresource is not a state of the image subresource itself, but is rather a property of how the data in memory is organized, and thus for each mechanism of accessing an image in the API the application **must** specify a parameter or structure member that indicates which image layout the image subresource(s) are considered to be in when the image will be accessed. For transfer commands, this is a parameter to the command (see [Clear Commands](#) and [Copy Commands](#)). For use as a framebuffer attachment, this is a member in the substructures of the `VkRenderPassCreateInfo` (see [Render Pass](#)). For use in a descriptor set, this is a member in the `VkDescriptorImageInfo` structure (see [Descriptor Set Updates](#)).

NOTE

`VK_IMAGE_LAYOUT_GENERAL` can be a useful catch-all image layout, but there are situations where a dedicated image layout must be used instead. Some examples include:

- `VK_IMAGE_LAYOUT_PRESENT_SRC_KHR`
- `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`

NOTE

While `VK_IMAGE_LAYOUT_GENERAL` suggests that all types of device access is possible, it does not mean that all patterns of memory accesses are safe in all situations. [Common Render Pass Data Races](#) outlines some situations where data races are unavoidable.

12.5.1. Image Layout Matching Rules

At the time that any command buffer command accessing an image executes on any queue, the layouts of the image subresources that are accessed **must** all match exactly the layout specified via the API controlling those accesses, except in case of accesses to an image with a depth/stencil format performed through descriptors referring to only a single aspect of the image, where the following relaxed matching rules apply:

- Descriptors referring just to the depth aspect of a depth/stencil image only need to match in the image layout of the depth aspect, thus `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL` and

`VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL` are considered to match.

- Descriptors referring just to the stencil aspect of a depth/stencil image only need to match in the image layout of the stencil aspect, thus `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL` and `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL` are considered to match.

When performing a layout transition on an image subresource, the old layout value **must** either equal the current layout of the image subresource (at the time the transition executes), or else be `VK_IMAGE_LAYOUT_UNDEFINED` (implying that the contents of the image subresource need not be preserved). The new layout used in a transition **must** not be any of:

- `VK_IMAGE_LAYOUT_UNDEFINED`
- `VK_IMAGE_LAYOUT_PREINITIALIZED`

The image layout of each image subresource of a depth/stencil image created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` is dependent on the last sample locations used to render to the image subresource as a depth/stencil attachment, thus applications **must** provide the same sample locations that were last used to render to the given image subresource whenever a layout transition of the image subresource happens, otherwise the contents of the depth aspect of the image subresource become undefined.

In addition, depth reads from a depth/stencil attachment referring to an image subresource range of a depth/stencil image created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` using different sample locations than what have been last used to perform depth writes to the image subresources of the same image subresource range return undefined values.

Similarly, depth writes to a depth/stencil attachment referring to an image subresource range of a depth/stencil image created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` using different sample locations than what have been last used to perform depth writes to the image subresources of the same image subresource range make the contents of the depth aspect of those image subresources undefined.

12.6. Image Views

Image objects are not directly accessed by pipeline shaders for reading or writing image data. Instead, *image views* representing contiguous ranges of the image subresources and containing additional metadata are used for that purpose. Views **must** be created on images of compatible types, and **must** represent a valid subset of image subresources.

Image views are represented by `VkImageView` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkImageView)
```

`VK_REMAINING_ARRAY_LAYERS` is a special constant value used for image views to indicate that all remaining array layers in an image after the base layer should be included in the view.

```
#define VK_REMAINING_ARRAY_LAYERS          (~0U)
```

`VK_REMAINING_MIP_LEVELS` is a special constant value used for image views to indicate that all remaining mipmap levels in an image after the base level should be included in the view.

```
#define VK_REMAINING_MIP_LEVELS          (~0U)
```

The types of image views that **can** be created are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageViewType {
    VK_IMAGE_VIEW_TYPE_1D = 0,
    VK_IMAGE_VIEW_TYPE_2D = 1,
    VK_IMAGE_VIEW_TYPE_3D = 2,
    VK_IMAGE_VIEW_TYPE_CUBE = 3,
    VK_IMAGE_VIEW_TYPE_1D_ARRAY = 4,
    VK_IMAGE_VIEW_TYPE_2D_ARRAY = 5,
    VK_IMAGE_VIEW_TYPE_CUBE_ARRAY = 6,
} VkImageViewType;
```

To create an image view, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateImageView(
    VkDevice                                device,
    const VkImageViewCreateInfo*            pCreateInfo,
    const VkAllocationCallbacks*           pAllocator,
    VkImageView*                            pView);
```

- `device` is the logical device that creates the image view.
- `pCreateInfo` is a pointer to a `VkImageViewCreateInfo` structure containing parameters to be used to create the image view.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pView` is a pointer to a `VkImageView` handle in which the resulting image view object is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateImageView` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateImageView-device-09667
`device` **must** support at least one queue family with one of the `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` capabilities

- VUID-vkCreateImageView-image-09179
`VkImageViewCreateInfo::image` **must** have been created from `device`
- VUID-vkCreateImageView-device-05068
The number of image views currently allocated from `device` plus 1 **must** be less than or equal to the total number of image views requested via `VkDeviceObjectReservationCreateInfo::imageViewRequestCount` specified when `device` was created
- VUID-vkCreateImageView-subresourceRange-05063
If `VkImageViewCreateInfo::subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS` and is greater than 1, or if `VkImageViewCreateInfo::subresourceRange.layerCount` is `VK_REMAINING_ARRAY_LAYERS` and the remaining number of layers in `VkImageViewCreateInfo::image` is greater than 1, the number of image views with more than one array layer currently allocated from `device` plus 1 **must** be less than or equal to the total number of image views requested via `VkDeviceObjectReservationCreateInfo::layeredImageViewRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateImageView-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkCreateImageView-pCreateInfo-parameter
`pCreateInfo` **must** be a valid pointer to a valid `VkImageViewCreateInfo` structure
- VUID-vkCreateImageView-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateImageView-pView-parameter
`pView` **must** be a valid pointer to a `VkImageView` handle
- VUID-vkCreateImageView-device-queuecount
The device **must** have been created with at least 1 queue

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkImageViewCreateInfo` structure is defined as:


```
// Provided by VK_VERSION_1_0
typedef struct VkImageViewCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkImageViewCreateFlags flags;
    VkImage               image;
    VkImageViewType       viewType;
    VkFormat              format;
    VkComponentMapping     components;
    VkImageSubresourceRange subresourceRange;
} VkImageViewCreateInfo;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **flags** is a bitmask of **VkImageViewCreateFlagBits** specifying additional parameters of the image view.
- **image** is a **VkImage** on which the view will be created.
- **viewType** is a **VkImageViewType** value specifying the type of the image view.
- **format** is a **VkFormat** specifying the format and type used to interpret texel blocks of the image.
- **components** is a **VkComponentMapping** structure specifying a remapping of color components (or of depth or stencil components after they have been converted into color components).
- **subresourceRange** is a **VkImageSubresourceRange** structure selecting the set of mipmap levels and array layers to be accessible to the view.

Some of the **image** creation parameters are inherited by the view. In particular, image view creation inherits the implicit parameter **usage** specifying the allowed usages of the image view that, by default, takes the value of the corresponding **usage** parameter specified in **VkImageCreateInfo** at image creation time. The implicit **usage** **can** be overridden by adding a **VkImageViewUsageCreateInfo** structure to the **pNext** chain, but the view usage **must** be a subset of the image usage. If **image** has a depth-stencil format and was created with a **VkImageStencilUsageCreateInfo** structure included in the **pNext** chain of **VkImageCreateInfo**, the usage is calculated based on the **subresource.aspectMask** provided:

- If **aspectMask** includes only **VK_IMAGE_ASPECT_STENCIL_BIT**, the implicit **usage** is equal to **VkImageStencilUsageCreateInfo::stencilUsage**.
- If **aspectMask** includes only **VK_IMAGE_ASPECT_DEPTH_BIT**, the implicit **usage** is equal to **VkImageCreateInfo::usage**.
- If both aspects are included in **aspectMask**, the implicit **usage** is equal to the intersection of **VkImageCreateInfo::usage** and **VkImageStencilUsageCreateInfo::stencilUsage**.

If **image** was created with the **VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT** flag, and if the **format** of the image is not **multi-planar format** **can** be different from the image's format, but if **image** was created without the **VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT** flag and they are not equal they **must** be *compatible*. Image format compatibility is defined in the **Format Compatibility Classes** section.

Views of compatible formats will have the same mapping between texel coordinates and memory locations irrespective of the **format**, with only the interpretation of the bit pattern changing.

If **image** was created with a **multi-planar format**, and the image view's **aspectMask** is one of **VK_IMAGE_ASPECT_PLANE_0_BIT**, **VK_IMAGE_ASPECT_PLANE_1_BIT** or **VK_IMAGE_ASPECT_PLANE_2_BIT**, the view's aspect mask is considered to be equivalent to **VK_IMAGE_ASPECT_COLOR_BIT** when used as a framebuffer attachment.

NOTE

Values intended to be used with one view format **may** not be exactly preserved when written or read through a different format. For example, an integer value that happens to have the bit pattern of a floating-point denorm or NaN **may** be flushed or canonicalized when written or read through a view with a floating-point format. Similarly, a value written through a signed normalized format that has a bit pattern exactly equal to -2^b **may** be changed to $-2^b + 1$ as described in [Conversion from Normalized Fixed-Point to Floating-Point](#).

If **image** was created with the **VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT** flag, **format** **must** be *compatible* with the image's format as described above; or **must** be an uncompressed format, in which case it **must** be *size-compatible* with the image's format. In this case, the resulting image view's texel dimensions equal the dimensions of the selected mip level divided by the compressed texel block size and rounded up.

The **VkComponentMapping** **components** member describes a remapping from components of the image to components of the vector returned by shader image instructions. This remapping **must** be the identity swizzle for any **VkImageView** used with a combined image sampler that enables **sampler Y'C_BC_R conversion**, input attachment descriptors, framebuffer attachments, and storage image descriptors.

If the image view is to be used with a sampler which supports **sampler Y'C_BC_R conversion**, an *identically defined object* of type **VkSamplerYcbcrConversion** to that used to create the sampler **must** be passed to **vkCreateImageView** in a **VkSamplerYcbcrConversionInfo** included in the **pNext** chain of **VkImageViewCreateInfo**. Conversely, if a **VkSamplerYcbcrConversion** object is passed to **vkCreateImageView**, an *identically defined* **VkSamplerYcbcrConversion** object **must** be used when sampling the image.

If the image has a **multi-planar format**, **subresourceRange.aspectMask** is **VK_IMAGE_ASPECT_COLOR_BIT**, and **usage** includes **VK_IMAGE_USAGE_SAMPLED_BIT**, then the **format** **must** be identical to the image **format** and the sampler to be used with the image view **must** enable **sampler Y'C_BC_R conversion**.

If **image** was created with the **VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT** and the image has a **multi-planar format**, and if **subresourceRange.aspectMask** is **VK_IMAGE_ASPECT_PLANE_0_BIT**, **VK_IMAGE_ASPECT_PLANE_1_BIT**, or **VK_IMAGE_ASPECT_PLANE_2_BIT**, **format** **must** be *compatible* with the corresponding plane of the image, and the sampler to be used with the image view **must** not enable **sampler Y'C_BC_R conversion**. The **width** and **height** of the single-plane image view **must** be derived from the multi-planar image's dimensions in the manner listed for **plane compatibility** for the plane.

Any view of an image plane will have the same mapping between texel coordinates and memory locations as used by the components of the color aspect, subject to the formulae relating texel

coordinates to lower-resolution planes as described in [Chroma Reconstruction](#). That is, if an R or B plane has a reduced resolution relative to the G plane of the multi-planar image, the image view operates using the (u_{plane}, v_{plane}) unnormalized coordinates of the reduced-resolution plane, and these coordinates access the same memory locations as the (u_{color}, v_{color}) unnormalized coordinates of the color aspect for which chroma reconstruction operations operate on the same (u_{plane}, v_{plane}) or (i_{plane}, j_{plane}) coordinates.

Table 13. Image Type and Image View Type Compatibility Requirements

Image View Type	Compatible Image Types
VK_IMAGE_VIEW_TYPE_1D	VK_IMAGE_TYPE_1D
VK_IMAGE_VIEW_TYPE_1D_ARRAY	VK_IMAGE_TYPE_1D
VK_IMAGE_VIEW_TYPE_2D	VK_IMAGE_TYPE_2D , VK_IMAGE_TYPE_3D
VK_IMAGE_VIEW_TYPE_2D_ARRAY	VK_IMAGE_TYPE_2D , VK_IMAGE_TYPE_3D
VK_IMAGE_VIEW_TYPE_CUBE	VK_IMAGE_TYPE_2D
VK_IMAGE_VIEW_TYPE_CUBE_ARRAY	VK_IMAGE_TYPE_2D
VK_IMAGE_VIEW_TYPE_3D	VK_IMAGE_TYPE_3D

Valid Usage

- VUID-VkImageViewCreateInfo-image-01003
If **image** was not created with **VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT** then **viewType** **must** not be **VK_IMAGE_VIEW_TYPE_CUBE** or **VK_IMAGE_VIEW_TYPE_CUBE_ARRAY**
- VUID-VkImageViewCreateInfo-viewType-01004
If the **imageCubeArray** feature is not enabled, **viewType** **must** not be **VK_IMAGE_VIEW_TYPE_CUBE_ARRAY**
- VUID-VkImageViewCreateInfo-image-06723
If **image** was created with **VK_IMAGE_TYPE_3D** but without **VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT** set then **viewType** **must** not be **VK_IMAGE_VIEW_TYPE_2D_ARRAY**
- VUID-VkImageViewCreateInfo-image-06727
If **image** was created with **VK_IMAGE_TYPE_3D** but without **VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT** set then **viewType** **must** not be **VK_IMAGE_VIEW_TYPE_2D**
- VUID-VkImageViewCreateInfo-image-04970
If **image** was created with **VK_IMAGE_TYPE_3D** and **viewType** is **VK_IMAGE_VIEW_TYPE_2D** or **VK_IMAGE_VIEW_TYPE_2D_ARRAY** then **subresourceRange.levelCount** **must** be 1
- VUID-VkImageViewCreateInfo-image-04972
If **image** was created with a **samples** value not equal to **VK_SAMPLE_COUNT_1_BIT** then **viewType** **must** be either **VK_IMAGE_VIEW_TYPE_2D** or **VK_IMAGE_VIEW_TYPE_2D_ARRAY**
- VUID-VkImageViewCreateInfo-image-04441
image **must** have been created with a **usage** value containing at least one of the usages defined in the [valid image usage](#) list for image views

- VUID-VkImageViewCreateInfo-None-02273
The **format features** of the resultant image view **must** contain at least one bit
- VUID-VkImageViewCreateInfo-usage-02274
If **usage** contains **VK_IMAGE_USAGE_SAMPLED_BIT**, then the **format features** of the resultant image view **must** contain **VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT**
- VUID-VkImageViewCreateInfo-usage-02275
If **usage** contains **VK_IMAGE_USAGE_STORAGE_BIT**, then the image view's **format features must** contain **VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT**
- VUID-VkImageViewCreateInfo-usage-02276
If **usage** contains **VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT**, then the image view's **format features must** contain **VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT**
- VUID-VkImageViewCreateInfo-usage-02277
If **usage** contains **VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT**, then the image view's **format features must** contain **VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT**
- VUID-VkImageViewCreateInfo-usage-08932
If **usage** contains **VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT**,

then the image view's **format features must** contain at least one of
VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT or
VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
- VUID-VkImageViewCreateInfo-subresourceRange-01478
subresourceRange.baseMipLevel must be less than the **mipLevels** specified in **VkImageCreateInfo** when **image** was created
- VUID-VkImageViewCreateInfo-subresourceRange-01718
If **subresourceRange.levelCount** is not **VK_REMAINING_MIP_LEVELS**, **subresourceRange.baseMipLevel + subresourceRange.levelCount must** be less than or equal to the **mipLevels** specified in **VkImageCreateInfo** when **image** was created
- VUID-VkImageViewCreateInfo-image-01482
If **image** is not a 3D image created with **VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT** set, or **viewType** is not **VK_IMAGE_VIEW_TYPE_2D** or **VK_IMAGE_VIEW_TYPE_2D_ARRAY**, **subresourceRange.baseArrayLayer must** be less than the **arrayLayers** specified in **VkImageCreateInfo** when **image** was created
- VUID-VkImageViewCreateInfo-subresourceRange-01483
If **subresourceRange.layerCount** is not **VK_REMAINING_ARRAY_LAYERS**, **image** is not a 3D image created with **VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT** set, or **viewType** is not **VK_IMAGE_VIEW_TYPE_2D** or **VK_IMAGE_VIEW_TYPE_2D_ARRAY**, **subresourceRange.layerCount must** be non-zero and **subresourceRange.baseArrayLayer + subresourceRange.layerCount must** be less than or equal to the **arrayLayers** specified in **VkImageCreateInfo** when **image** was created
- VUID-VkImageViewCreateInfo-image-02724
If **image** is a 3D image created with **VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT** set, and **viewType** is **VK_IMAGE_VIEW_TYPE_2D** or **VK_IMAGE_VIEW_TYPE_2D_ARRAY**, **subresourceRange.baseArrayLayer must** be less than the depth computed from **baseMipLevel** and **extent.depth** specified in **VkImageCreateInfo** when **image** was created, according to

the formula defined in [Image Mip Level Sizing](#)

- VUID-VkImageViewCreateInfo-subresourceRange-02725

If `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `image` is a 3D image created with `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT` set, and `viewType` is `VK_IMAGE_VIEW_TYPE_2D` or `VK_IMAGE_VIEW_TYPE_2D_ARRAY`, `subresourceRange.layerCount` **must** be non-zero and `subresourceRange.baseArrayLayer + subresourceRange.layerCount` **must** be less than or equal to the depth computed from `baseMipLevel` and `extent.depth` specified in `VkImageCreateInfo` when `image` was created, according to the formula defined in [Image Mip Level Sizing](#)

- VUID-VkImageViewCreateInfo-image-01761

If `image` was created with the `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` flag, but without the `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT` flag, and if the `format` of the `image` is not a [multi-planar format](#), `format` **must** be compatible with the `format` used to create `image`, as defined in [Format Compatibility Classes](#)

- VUID-VkImageViewCreateInfo-image-01583

If `image` was created with the `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT` flag, `format` **must** be compatible with, or **must** be an uncompressed format that is [size-compatible](#) with, the `format` used to create `image`

- VUID-VkImageViewCreateInfo-image-07072

If `image` was created with the `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT` flag and `format` is a non-compressed format, the `levelCount` member of `subresourceRange` **must** be 1

- VUID-VkImageViewCreateInfo-image-09487

If `image` was created with the `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT` flag, and `format` is a non-compressed format, then the `layerCount` member of `subresourceRange` **must** be 1

- VUID-VkImageViewCreateInfo-pNext-01585

If a `VkImageFormatListCreateInfo` structure was included in the `pNext` chain of the `VkImageCreateInfo` structure used when creating `image` and `VkImageFormatListCreateInfo::viewFormatCount` is not zero then `format` **must** be one of the formats in `VkImageFormatListCreateInfo::pViewFormats`

- VUID-VkImageViewCreateInfo-image-01586

If `image` was created with the `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` flag, if the `format` of the `image` is a [multi-planar format](#), and if `subresourceRange.aspectMask` is one of the [multi-planar aspect mask](#) bits, then `format` **must** be compatible with the `VkFormat` for the plane of the `image format` indicated by `subresourceRange.aspectMask`, as defined in [Compatible Formats of Planes of Multi-Planar Formats](#)

- VUID-VkImageViewCreateInfo-subresourceRange-07818

`subresourceRange.aspectMask` **must** only have at most 1 valid [multi-planar aspect mask](#) bit

- VUID-VkImageViewCreateInfo-image-01762

If `image` was not created with the `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` flag, or if the `format` of the `image` is a [multi-planar format](#) and if `subresourceRange.aspectMask` is `VK_IMAGE_ASPECT_COLOR_BIT`, `format` **must** be identical to the `format` used to create `image`

- VUID-VkImageViewCreateInfo-format-06415

If the image view [requires a sampler Y'C_BR conversion](#) and `usage` contains

VK_IMAGE_USAGE_SAMPLED_BIT, then the pNext chain **must** include a VkSamplerYcbcrConversionInfo structure with a conversion value other than VK_NULL_HANDLE

- VUID-VkImageViewCreateInfo-format-04714

If format has a _422 or _420 suffix then image **must** have been created with a width that is a multiple of 2

- VUID-VkImageViewCreateInfo-format-04715

If format has a _420 suffix then image **must** have been created with a height that is a multiple of 2

- VUID-VkImageViewCreateInfo-pNext-01970

If the pNext chain includes a VkSamplerYcbcrConversionInfo structure with a conversion value other than VK_NULL_HANDLE, all members of components **must** have the identity swizzle

- VUID-VkImageViewCreateInfo-pNext-06658

If the pNext chain includes a VkSamplerYcbcrConversionInfo structure with a conversion value other than VK_NULL_HANDLE, format **must** be the same used in VkSamplerYcbcrConversionCreateInfo::format

- VUID-VkImageViewCreateInfo-image-01020

If image is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object

- VUID-VkImageViewCreateInfo-subResourceRange-01021

viewType **must** be compatible with the type of image as shown in the view type compatibility table

- VUID-VkImageViewCreateInfo-image-08957

If image has an QNX Screen external format, format **must** be VK_FORMAT_UNDEFINED

- VUID-VkImageViewCreateInfo-image-08958

If image has an QNX Screen external format, the pNext chain **must** include a VkSamplerYcbcrConversionInfo structure with a conversion object created with the same external format as image

- VUID-VkImageViewCreateInfo-image-08959

If image has an QNX Screen external format, all members of components **must** be the identity swizzle

- VUID-VkImageViewCreateInfo-image-02086

If image was created with usage containing VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR, viewType **must** be VK_IMAGE_VIEW_TYPE_2D or VK_IMAGE_VIEW_TYPE_2D_ARRAY

- VUID-VkImageViewCreateInfo-usage-04550

If the attachmentFragmentShadingRate feature is enabled, and the usage for the image view includes VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR, then the image view's format features **must** contain VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR

- VUID-VkImageViewCreateInfo-usage-04551

If the attachmentFragmentShadingRate feature is enabled, the usage for the image view

includes `VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`, and `layeredShadingRateAttachments` is `VK_FALSE`, `subresourceRange.layerCount` **must** be 1

- VUID-VkImageViewCreateInfo-pNext-02662

If the `pNext` chain includes a `VkImageViewUsageCreateInfo` structure, and `image` was not created with a `VkImageStencilUsageCreateInfo` structure included in the `pNext` chain of `VkImageCreateInfo`, its `usage` member **must** not include any bits that were not set in the `usage` member of the `VkImageCreateInfo` structure used to create `image`

- VUID-VkImageViewCreateInfo-pNext-02663

If the `pNext` chain includes a `VkImageViewUsageCreateInfo` structure, `image` was created with a `VkImageStencilUsageCreateInfo` structure included in the `pNext` chain of `VkImageCreateInfo`, and `subresourceRange.aspectMask` includes `VK_IMAGE_ASPECT_STENCIL_BIT`, the `usage` member of the `VkImageViewUsageCreateInfo` structure **must** not include any bits that were not set in the `usage` member of the `VkImageStencilUsageCreateInfo` structure used to create `image`

- VUID-VkImageViewCreateInfo-pNext-02664

If the `pNext` chain includes a `VkImageViewUsageCreateInfo` structure, `image` was created with a `VkImageStencilUsageCreateInfo` structure included in the `pNext` chain of `VkImageCreateInfo`, and `subresourceRange.aspectMask` includes bits other than `VK_IMAGE_ASPECT_STENCIL_BIT`, the `usage` member of the `VkImageViewUsageCreateInfo` structure **must** not include any bits that were not set in the `usage` member of the `VkImageCreateInfo` structure used to create `image`

- VUID-VkImageViewCreateInfo-imageViewType-04973

If `viewType` is `VK_IMAGE_VIEW_TYPE_1D`, `VK_IMAGE_VIEW_TYPE_2D`, or `VK_IMAGE_VIEW_TYPE_3D`; and `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, then `subresourceRange.layerCount` **must** be 1

- VUID-VkImageViewCreateInfo-imageViewType-04974

If `viewType` is `VK_IMAGE_VIEW_TYPE_1D`, `VK_IMAGE_VIEW_TYPE_2D`, or `VK_IMAGE_VIEW_TYPE_3D`; and `subresourceRange.layerCount` is `VK_REMAINING_ARRAY_LAYERS`, then the remaining number of layers **must** be 1

- VUID-VkImageViewCreateInfo-viewType-02960

If `viewType` is `VK_IMAGE_VIEW_TYPE_CUBE` and `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `subresourceRange.layerCount` **must** be 6

- VUID-VkImageViewCreateInfo-viewType-02961

If `viewType` is `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY` and `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `subresourceRange.layerCount` **must** be a multiple of 6

- VUID-VkImageViewCreateInfo-viewType-02962

If `viewType` is `VK_IMAGE_VIEW_TYPE_CUBE` and `subresourceRange.layerCount` is `VK_REMAINING_ARRAY_LAYERS`, the remaining number of layers **must** be 6

- VUID-VkImageViewCreateInfo-viewType-02963

If `viewType` is `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY` and `subresourceRange.layerCount` is `VK_REMAINING_ARRAY_LAYERS`, the remaining number of layers **must** be a multiple of 6

- VUID-VkImageViewCreateInfo-subresourceRange-05064

If `subresourceRange.levelCount` is not `VK_REMAINING_MIP_LEVELS`, `subresourceRange.levelCount` **must** be less than or equal to

VkDeviceObjectReservationCreateInfo::maxImageViewMipLevels

- VUID-VkImageViewCreateInfo-subresourceRange-05200

If `subresourceRange.levelCount` is `VK_REMAINING_MIP_LEVELS`, the remaining number of mip levels **must** be less than or equal to `VkDeviceObjectReservationCreateInfo::maxImageViewMipLevels`

- VUID-VkImageViewCreateInfo-subresourceRange-05065

If `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `subresourceRange.layerCount` **must** be less than or equal to `VkDeviceObjectReservationCreateInfo::maxImageViewArrayLayers`

- VUID-VkImageViewCreateInfo-subresourceRange-05201

If `subresourceRange.layerCount` is `VK_REMAINING_ARRAY_LAYERS`, the remaining number of layers **must** be less than or equal to `VkDeviceObjectReservationCreateInfo::maxImageViewMipLevels`

- VUID-VkImageViewCreateInfo-subresourceRange-05066

If `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS` and is greater than 1, or if `subresourceRange.layerCount` is `VK_REMAINING_ARRAY_LAYERS` and the remaining number of layers is greater than 1, then if `subresourceRange.levelCount` is not `VK_REMAINING_MIP_LEVELS`, `subresourceRange.levelCount` **must** be less than or equal to `VkDeviceObjectReservationCreateInfo::maxLayeredImageViewMipLevels`

- VUID-VkImageViewCreateInfo-subresourceRange-05202

If `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS` and is greater than 1, or if `subresourceRange.layerCount` is `VK_REMAINING_ARRAY_LAYERS` and the remaining number of layers is greater than 1, then if `subresourceRange.levelCount` is `VK_REMAINING_MIP_LEVELS`, the remaining number of mip levels **must** be less than or equal to `VkDeviceObjectReservationCreateInfo::maxLayeredImageViewMipLevels`

- VUID-VkImageViewCreateInfo-subresourceRange-09594

`subresourceRange.aspectMask` **must** be valid for the format the image was created with

Valid Usage (Implicit)

- VUID-VkImageViewCreateInfo-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO`

- VUID-VkImageViewCreateInfo-pNext-pNext

Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkImageViewASTCDecodeModeEXT`, `VkImageViewUsageCreateInfo`, or `VkSamplerYcbcrConversionInfo`

- VUID-VkImageViewCreateInfo-sType-unique

The `sType` value of each structure in the `pNext` chain **must** be unique

- VUID-VkImageViewCreateInfo-flags-zeroBitmask

`flags` **must** be 0

- VUID-VkImageViewCreateInfo-image-parameter

`image` **must** be a valid `VkImage` handle

- VUID-VkImageViewCreateInfo-viewType-parameter
`viewType` **must** be a valid `VkImageViewType` value
- VUID-VkImageViewCreateInfo-format-parameter
`format` **must** be a valid `VkFormat` value
- VUID-VkImageViewCreateInfo-components-parameter
`components` **must** be a valid `VkComponentMapping` structure
- VUID-VkImageViewCreateInfo-subresourceRange-parameter
`subresourceRange` **must** be a valid `VkImageSubresourceRange` structure

Bits which **can** be set in `VkImageViewCreateInfo::flags`, specifying additional parameters of an image view, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageViewCreateFlagBits {
} VkImageViewCreateFlagBits;
```

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkImageViewCreateFlags;
```

`VkImageViewCreateFlags` is a bitmask type for setting a mask of zero or more `VkImageViewCreateFlagBits`.

The set of usages for the created image view **can** be restricted compared to the parent image's `usage` flags by adding a `VkImageViewUsageCreateInfo` structure to the `pNext` chain of `VkImageViewCreateInfo`.

The `VkImageViewUsageCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkImageViewUsageCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkImageUsageFlags  usage;
} VkImageViewUsageCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `usage` is a bitmask of `VkImageUsageFlagBits` specifying allowed usages of the image view.

When this structure is chained to `VkImageViewCreateInfo` the `usage` field overrides the implicit `usage` parameter inherited from image creation time and its value is used instead for the purposes of determining the valid usage conditions of `VkImageViewCreateInfo`.

Valid Usage (Implicit)

- VUID-VkImageViewUsageCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_VIEW_USAGE_CREATE_INFO`
- VUID-VkImageViewUsageCreateInfo-usage-parameter
`usage` **must** be a valid combination of `VkImageUsageFlagBits` values
- VUID-VkImageViewUsageCreateInfo-usage-requiredbitmask
`usage` **must** not be `0`

The `VkImageSubresourceRange` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageSubresourceRange {
    VkImageAspectFlags    aspectMask;
    uint32_t              baseMipLevel;
    uint32_t              levelCount;
    uint32_t              baseArrayLayer;
    uint32_t              layerCount;
} VkImageSubresourceRange;
```

- `aspectMask` is a bitmask of `VkImageAspectFlagBits` specifying which aspect(s) of the image are included in the view.
- `baseMipLevel` is the first mipmap level accessible to the view.
- `levelCount` is the number of mipmap levels (starting from `baseMipLevel`) accessible to the view.
- `baseArrayLayer` is the first array layer accessible to the view.
- `layerCount` is the number of array layers (starting from `baseArrayLayer`) accessible to the view.

The number of mipmap levels and array layers **must** be a subset of the image subresources in the image. If an application wants to use all mip levels or layers in an image after the `baseMipLevel` or `baseArrayLayer`, it **can** set `levelCount` and `layerCount` to the special values `VK_REMAINING_MIP_LEVELS` and `VK_REMAINING_ARRAY_LAYERS` without knowing the exact number of mip levels or layers.

For cube and cube array image views, the layers of the image view starting at `baseArrayLayer` correspond to faces in the order +X, -X, +Y, -Y, +Z, -Z. For cube arrays, each set of six sequential layers is a single cube, so the number of cube maps in a cube map array view is `layerCount / 6`, and image array layer (`baseArrayLayer + i`) is face index (`i mod 6`) of cube `i / 6`. If the number of layers in the view, whether set explicitly in `layerCount` or implied by `VK_REMAINING_ARRAY_LAYERS`, is not a multiple of 6, the last cube map in the array **must** not be accessed.

`aspectMask` **must** be only `VK_IMAGE_ASPECT_COLOR_BIT`, `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT` if `format` is a color, depth-only or stencil-only format, respectively, except if `format` is a `multi-planar format`. If using a depth/stencil format with both depth and stencil components, `aspectMask` **must** include at least one of `VK_IMAGE_ASPECT_DEPTH_BIT` and `VK_IMAGE_ASPECT_STENCIL_BIT`, and **can** include both.

When the `VkImageSubresourceRange` structure is used to select a subset of the slices of a 3D image's mip level in order to create a 2D or 2D array image view of a 3D image created with `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT`, `baseArrayLayer` and `layerCount` specify the first slice index and the number of slices to include in the created image view. Such an image view **can** be used as a framebuffer attachment that refers only to the specified range of slices of the selected mip level. However, any layout transitions performed on such an attachment view during a render pass instance still apply to the entire subresource referenced which includes all the slices of the selected mip level.

When using an image view of a depth/stencil image to populate a descriptor set (e.g. for sampling in the shader, or for use as an input attachment), the `aspectMask` **must** only include one bit, which selects whether the image view is used for depth reads (i.e. using a floating-point sampler or input attachment in the shader) or stencil reads (i.e. using an unsigned integer sampler or input attachment in the shader). When an image view of a depth/stencil image is used as a depth/stencil framebuffer attachment, the `aspectMask` is ignored and both depth and stencil image subresources are used.

When creating a `VkImageView`, if sampler Y'_BC_R conversion is enabled in the sampler, the `aspectMask` of a `subresourceRange` used by the `VkImageView` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`.

When creating a `VkImageView`, if sampler Y'_BC_R conversion is not enabled in the sampler and the image `format` is `multi-planar format`, the image **must** have been created with `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`, and the `aspectMask` of the `VkImageView`'s `subresourceRange` **must** be `VK_IMAGE_ASPECT_PLANE_0_BIT`, `VK_IMAGE_ASPECT_PLANE_1_BIT` or `VK_IMAGE_ASPECT_PLANE_2_BIT`.

Valid Usage

- VUID-VkImageSubresourceRange-levelCount-01720
If `levelCount` is not `VK_REMAINING_MIP_LEVELS`, it **must** be greater than 0
- VUID-VkImageSubresourceRange-layerCount-01721
If `layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, it **must** be greater than 0
- VUID-VkImageSubresourceRange-aspectMask-01670
If `aspectMask` includes `VK_IMAGE_ASPECT_COLOR_BIT`, then it **must** not include any of `VK_IMAGE_ASPECT_PLANE_0_BIT`, `VK_IMAGE_ASPECT_PLANE_1_BIT`, or `VK_IMAGE_ASPECT_PLANE_2_BIT`
- VUID-VkImageSubresourceRange-aspectMask-02278
`aspectMask` **must** not include `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT` for any index *i*

Valid Usage (Implicit)

- VUID-VkImageSubresourceRange-aspectMask-parameter
`aspectMask` **must** be a valid combination of `VkImageAspectFlagBits` values
- VUID-VkImageSubresourceRange-aspectMask-requiredbitmask
`aspectMask` **must** not be 0

Bits which **can** be set in an aspect mask to specify aspects of an image for purposes such as

identifying a subresource, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageAspectFlagBits {
    VK_IMAGE_ASPECT_COLOR_BIT = 0x00000001,
    VK_IMAGE_ASPECT_DEPTH_BIT = 0x00000002,
    VK_IMAGE_ASPECT_STENCIL_BIT = 0x00000004,
    VK_IMAGE_ASPECT_METADATA_BIT = 0x00000008,
    // Provided by VK_VERSION_1_1
    VK_IMAGE_ASPECT_PLANE_0_BIT = 0x00000010,
    // Provided by VK_VERSION_1_1
    VK_IMAGE_ASPECT_PLANE_1_BIT = 0x00000020,
    // Provided by VK_VERSION_1_1
    VK_IMAGE_ASPECT_PLANE_2_BIT = 0x00000040,
    // Provided by VK_EXT_image_drm_format_modifier
    VK_IMAGE_ASPECT_MEMORY_PLANE_0_BIT_EXT = 0x00000080,
    // Provided by VK_EXT_image_drm_format_modifier
    VK_IMAGE_ASPECT_MEMORY_PLANE_1_BIT_EXT = 0x00000100,
    // Provided by VK_EXT_image_drm_format_modifier
    VK_IMAGE_ASPECT_MEMORY_PLANE_2_BIT_EXT = 0x00000200,
    // Provided by VK_EXT_image_drm_format_modifier
    VK_IMAGE_ASPECT_MEMORY_PLANE_3_BIT_EXT = 0x00000400,
} VkImageAspectFlagBits;
```

- `VK_IMAGE_ASPECT_COLOR_BIT` specifies the color aspect.
- `VK_IMAGE_ASPECT_DEPTH_BIT` specifies the depth aspect.
- `VK_IMAGE_ASPECT_STENCIL_BIT` specifies the stencil aspect.
- `VK_IMAGE_ASPECT_METADATA_BIT` specifies the metadata aspect used for [sparse resource](#) operations.
- `VK_IMAGE_ASPECT_PLANE_0_BIT` specifies plane 0 of a *multi-planar* image format.
- `VK_IMAGE_ASPECT_PLANE_1_BIT` specifies plane 1 of a *multi-planar* image format.
- `VK_IMAGE_ASPECT_PLANE_2_BIT` specifies plane 2 of a *multi-planar* image format.
- `VK_IMAGE_ASPECT_MEMORY_PLANE_0_BIT_EXT` specifies *memory plane 0*.
- `VK_IMAGE_ASPECT_MEMORY_PLANE_1_BIT_EXT` specifies *memory plane 1*.
- `VK_IMAGE_ASPECT_MEMORY_PLANE_2_BIT_EXT` specifies *memory plane 2*.
- `VK_IMAGE_ASPECT_MEMORY_PLANE_3_BIT_EXT` specifies *memory plane 3*.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkImageAspectFlags;
```

`VkImageAspectFlags` is a bitmask type for setting a mask of zero or more `VkImageAspectFlagBits`.

The `VkComponentMapping` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkComponentMapping {
    VkComponentSwizzle    r;
    VkComponentSwizzle    g;
    VkComponentSwizzle    b;
    VkComponentSwizzle    a;
} VkComponentMapping;
```

- **r** is a [VkComponentSwizzle](#) specifying the component value placed in the R component of the output vector.
- **g** is a [VkComponentSwizzle](#) specifying the component value placed in the G component of the output vector.
- **b** is a [VkComponentSwizzle](#) specifying the component value placed in the B component of the output vector.
- **a** is a [VkComponentSwizzle](#) specifying the component value placed in the A component of the output vector.

Valid Usage (Implicit)

- VUID-VkComponentMapping-r-parameter
r must be a valid [VkComponentSwizzle](#) value
- VUID-VkComponentMapping-g-parameter
g must be a valid [VkComponentSwizzle](#) value
- VUID-VkComponentMapping-b-parameter
b must be a valid [VkComponentSwizzle](#) value
- VUID-VkComponentMapping-a-parameter
a must be a valid [VkComponentSwizzle](#) value

Possible values of the members of [VkComponentMapping](#), specifying the component values placed in each component of the output vector, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkComponentSwizzle {
    VK_COMPONENT_SWIZZLE_IDENTITY = 0,
    VK_COMPONENT_SWIZZLE_ZERO = 1,
    VK_COMPONENT_SWIZZLE_ONE = 2,
    VK_COMPONENT_SWIZZLE_R = 3,
    VK_COMPONENT_SWIZZLE_G = 4,
    VK_COMPONENT_SWIZZLE_B = 5,
    VK_COMPONENT_SWIZZLE_A = 6,
} VkComponentSwizzle;
```

- **VK_COMPONENT_SWIZZLE_IDENTITY** specifies that the component is set to the identity swizzle.

- `VK_COMPONENT_SWIZZLE_ZERO` specifies that the component is set to zero.
- `VK_COMPONENT_SWIZZLE_ONE` specifies that the component is set to either 1 or 1.0, depending on whether the type of the image view format is integer or floating-point respectively, as determined by the [Format Definition](#) section for each `VkFormat`.
- `VK_COMPONENT_SWIZZLE_R` specifies that the component is set to the value of the R component of the image.
- `VK_COMPONENT_SWIZZLE_G` specifies that the component is set to the value of the G component of the image.
- `VK_COMPONENT_SWIZZLE_B` specifies that the component is set to the value of the B component of the image.
- `VK_COMPONENT_SWIZZLE_A` specifies that the component is set to the value of the A component of the image.

Setting the identity swizzle on a component is equivalent to setting the identity mapping on that component. That is:

Table 14. Component Mappings Equivalent To `VK_COMPONENT_SWIZZLE_IDENTITY`

Component	Identity Mapping
<code>components.r</code>	<code>VK_COMPONENT_SWIZZLE_R</code>
<code>components.g</code>	<code>VK_COMPONENT_SWIZZLE_G</code>
<code>components.b</code>	<code>VK_COMPONENT_SWIZZLE_B</code>
<code>components.a</code>	<code>VK_COMPONENT_SWIZZLE_A</code>

If the `pNext` chain includes a `VkImageViewASTCDecodeModeEXT` structure, then that structure includes a parameter specifying the decode mode for image views using ASTC compressed formats.

The `VkImageViewASTCDecodeModeEXT` structure is defined as:

```
// Provided by VK_EXT_astc_decode_mode
typedef struct VkImageViewASTCDecodeModeEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkFormat            decodeMode;
} VkImageViewASTCDecodeModeEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `decodeMode` is the intermediate format used to decode ASTC compressed formats.

Valid Usage

- VUID-VkImageViewASTCDecodeModeEXT-decodeMode-02230
`decodeMode` **must** be one of `VK_FORMAT_R16G16B16A16_SFLOAT`, `VK_FORMAT_R8G8B8A8_UNORM`, or `VK_FORMAT_E5B9G9R9_UFLOAT_PACK32`

- VUID-VkImageViewASTCDecodeModeEXT-decodeMode-02231
If the `decodeModeSharedExponent` feature is not enabled, `decodeMode` **must** not be `VK_FORMAT_E5B9G9R9_UFLOAT_PACK32`
- VUID-VkImageViewASTCDecodeModeEXT-decodeMode-02232
If `decodeMode` is `VK_FORMAT_R8G8B8A8_UNORM` the image view **must** not include blocks using any of the ASTC HDR modes
- VUID-VkImageViewASTCDecodeModeEXT-format-04084
`format` of the image view **must** be one of the [ASTC Compressed Image Formats](#)

If `format` uses sRGB encoding then the `decodeMode` has no effect.

Valid Usage (Implicit)

- VUID-VkImageViewASTCDecodeModeEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_VIEW_ASTC_DECODE_MODE_EXT`
- VUID-VkImageViewASTCDecodeModeEXT-decodeMode-parameter
`decodeMode` **must** be a valid [VkFormat](#) value

To destroy an image view, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyImageView(
    VkDevice          device,
    VkImageView       imageView,
    const VkAllocationCallbacks* pAllocator);
```

- `device` is the logical device that destroys the image view.
- `imageView` is the image view to destroy.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage

- VUID-vkDestroyImageView-imageView-01026
All submitted commands that refer to `imageView` **must** have completed execution

Valid Usage (Implicit)

- VUID-vkDestroyImageView-device-parameter
`device` **must** be a valid [VkDevice](#) handle
- VUID-vkDestroyImageView-imageView-parameter
If `imageView` is not `VK_NULL_HANDLE`, `imageView` **must** be a valid [VkImageView](#) handle
- VUID-vkDestroyImageView-pAllocator-null

`pAllocator` must be `NULL`

- `VUID-vkDestroyImageView-imageView-parent`

If `imageView` is a valid handle, it **must** have been created, allocated, or retrieved from device

Host Synchronization

- Host access to `imageView` **must** be externally synchronized

12.6.1. Image View Format Features

Valid uses of a `VkImageView` may depend on the image view's *format features*, defined below. Such constraints are documented in the affected valid usage statement.

- If `VkImageViewCreateInfo::image` was created with `VK_IMAGE_TILING_LINEAR`, then the image view's set of *format features* is the value of `VkFormatProperties::linearTilingFeatures` found by calling `vkGetPhysicalDeviceFormatProperties` on the same `format` as `VkImageViewCreateInfo::format`.
- If `VkImageViewCreateInfo::image` was created with `VK_IMAGE_TILING_OPTIMAL`, or a QNX Screen buffer external format, then the image view's set of *format features* is the value of `VkFormatProperties::optimalTilingFeatures` found by calling `vkGetPhysicalDeviceFormatProperties` on the same `format` as `VkImageViewCreateInfo::format`.
- If `VkImageViewCreateInfo::image` was created with a QNX Screen buffer external format, then the image view's set of *format features* is the value of `VkScreenBufferFormatPropertiesQNX::formatFeatures` found by calling `vkGetScreenBufferPropertiesQNX` on the QNX Screen buffer that was imported to the `VkDeviceMemory` to which the `VkImageViewCreateInfo::image` is bound.
- If `VkImageViewCreateInfo::image` was created with `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then:
 - The image's DRM format modifier is the value of `VkImageDrmFormatModifierPropertiesEXT::drmFormatModifier` found by calling `vkGetImageDrmFormatModifierPropertiesEXT`.
 - Let `VkDrmFormatModifierPropertiesListEXT::pDrmFormatModifierProperties` be the array found by calling `vkGetPhysicalDeviceFormatProperties2` on the same `format` as `VkImageViewCreateInfo::format`.
 - Let `VkDrmFormatModifierPropertiesEXT prop` be the array element whose `drmFormatModifier` member is the value of the image's DRM format modifier.
 - Then the image view's set of *format features* is `prop::drmFormatModifierTilingFeatures`.

12.7. Resource Memory Association

Resources are initially created as *virtual allocations* with no backing memory. Device memory is allocated separately (see [Device Memory](#)) and then associated with the resource. This association is

done differently for sparse and non-sparse resources.

Resources created with any of the sparse creation flags are considered sparse resources. Resources created without these flags are non-sparse. The details on resource memory association for sparse resources is described in [Sparse Resources](#).

Non-sparse resources **must** be bound completely and contiguously to a single `VkDeviceMemory` object before the resource is passed as a parameter to any of the following operations:

- creating image or buffer views
- updating descriptor sets
- recording commands in a command buffer

Once bound, the memory binding is immutable for the lifetime of the resource.

In a logical device representing more than one physical device, buffer and image resources exist on all physical devices but **can** be bound to memory differently on each. Each such replicated resource is an *instance* of the resource. For sparse resources, each instance **can** be bound to memory arbitrarily differently. For non-sparse resources, each instance **can** either be bound to the local or a peer instance of the memory, or for images **can** be bound to rectangular regions from the local and/or peer instances. When a resource is used in a descriptor set, each physical device interprets the descriptor according to its own instance's binding to memory.

NOTE

There are no new copy commands to transfer data between physical devices. Instead, an application **can** create a resource with a peer mapping and use it as the source or destination of a transfer command executed by a single physical device to copy the data from one physical device to another.

To determine the memory requirements for a buffer resource, call:

```
// Provided by VK_VERSION_1_0
void vkGetBufferMemoryRequirements(
    VkDevice          device,
    VkBuffer          buffer,
    VkMemoryRequirements* pMemoryRequirements);
```

- `device` is the logical device that owns the buffer.
- `buffer` is the buffer to query.
- `pMemoryRequirements` is a pointer to a `VkMemoryRequirements` structure in which the memory requirements of the buffer object are returned.

Valid Usage (Implicit)

- VUID-vkGetBufferMemoryRequirements-device-parameter `device` **must** be a valid `VkDevice` handle
- VUID-vkGetBufferMemoryRequirements-buffer-parameter

buffer **must** be a valid [VkBuffer](#) handle

- VUID-vkGetBufferMemoryRequirements-pMemoryRequirements-parameter
pMemoryRequirements **must** be a valid pointer to a [VkMemoryRequirements](#) structure
- VUID-vkGetBufferMemoryRequirements-buffer-parent
buffer **must** have been created, allocated, or retrieved from **device**

To determine the memory requirements for an image resource which is not created with the **VK_IMAGE_CREATE_DISJOINT_BIT** flag set, call:

```
// Provided by VK_VERSION_1_0
void vkGetImageMemoryRequirements(
    VkDevice          device,
    VkImage           image,
    VkMemoryRequirements* pMemoryRequirements);
```

- **device** is the logical device that owns the image.
- **image** is the image to query.
- **pMemoryRequirements** is a pointer to a [VkMemoryRequirements](#) structure in which the memory requirements of the image object are returned.

Valid Usage

- VUID-vkGetImageMemoryRequirements-image-01588
image **must** not have been created with the **VK_IMAGE_CREATE_DISJOINT_BIT** flag set
- VUID-vkGetImageMemoryRequirements-image-08960
If **image** was created with the **VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX** external memory handle type, then **image** **must** be bound to memory

Valid Usage (Implicit)

- VUID-vkGetImageMemoryRequirements-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkGetImageMemoryRequirements-image-parameter
image **must** be a valid [VkImage](#) handle
- VUID-vkGetImageMemoryRequirements-pMemoryRequirements-parameter
pMemoryRequirements **must** be a valid pointer to a [VkMemoryRequirements](#) structure
- VUID-vkGetImageMemoryRequirements-image-parent
image **must** have been created, allocated, or retrieved from **device**

The [VkMemoryRequirements](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkMemoryRequirements {
    VkDeviceSize    size;
    VkDeviceSize    alignment;
    uint32_t        memoryTypeBits;
} VkMemoryRequirements;
```

- **size** is the size, in bytes, of the memory allocation **required** for the resource.
- **alignment** is the alignment, in bytes, of the offset within the allocation **required** for the resource.
- **memoryTypeBits** is a bitmask and contains one bit set for every supported memory type for the resource. Bit **i** is set if and only if the memory type **i** in the `VkPhysicalDeviceMemoryProperties` structure for the physical device is supported for the resource.

The implementation guarantees certain properties about the memory requirements returned by `vkGetBufferMemoryRequirements` and `vkGetImageMemoryRequirements`:

- The **memoryTypeBits** member always contains at least one bit set.
- If **buffer** is a `VkBuffer` not created with the `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` or `VK_BUFFER_CREATE_PROTECTED_BIT` bits set, or if **image** is a **linear** image that was not created with the `VK_IMAGE_CREATE_PROTECTED_BIT` bit set, then the **memoryTypeBits** member always contains at least one bit set corresponding to a `VkMemoryType` with a **propertyFlags** that has both the `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT` bit and the `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` bit set. In other words, mappable coherent memory **can** always be attached to these objects.
- If **buffer** was created with `VkExternalMemoryBufferCreateInfo::handleTypes` set to 0 or **image** was created with `VkExternalMemoryImageCreateInfo::handleTypes` set to 0, the **memoryTypeBits** member always contains at least one bit set corresponding to a `VkMemoryType` with a **propertyFlags** that has the `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT` bit set.
- The **memoryTypeBits** member is identical for all `VkBuffer` objects created with the same value for the **flags** and **usage** members in the `VkBufferCreateInfo` structure and the **handleTypes** member of the `VkExternalMemoryBufferCreateInfo` structure passed to `vkCreateBuffer`. Further, if **usage1** and **usage2** of type `VkBufferUsageFlags` are such that the bits set in **usage2** are a subset of the bits set in **usage1**, and they have the same **flags** and `VkExternalMemoryBufferCreateInfo::handleTypes`, then the bits set in **memoryTypeBits** returned for **usage1** **must** be a subset of the bits set in **memoryTypeBits** returned for **usage2**, for all values of **flags**.
- The **alignment** member is a power of two.
- The **alignment** member is identical for all `VkBuffer` objects created with the same combination of values for the **usage** and **flags** members in the `VkBufferCreateInfo` structure passed to `vkCreateBuffer`.
- The **alignment** member satisfies the buffer descriptor offset alignment requirements associated with the `VkBuffer`'s **usage**:
 - If **usage** included `VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT` or `VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT`, **alignment** **must** be an integer multiple of `VkPhysicalDeviceLimits::minTexelBufferOffsetAlignment`.

- If `usage` included `VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT`, `alignment` **must** be an integer multiple of `VkPhysicalDeviceLimits::minUniformBufferOffsetAlignment`.
- If `usage` included `VK_BUFFER_USAGE_STORAGE_BUFFER_BIT`, `alignment` **must** be an integer multiple of `VkPhysicalDeviceLimits::minStorageBufferOffsetAlignment`.
- For images created with a color format, the `memoryTypeBits` member is identical for all `VkImage` objects created with the same combination of values for the `tiling` member, the `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` bit and `VK_IMAGE_CREATE_PROTECTED_BIT` bit of the `flags` member, the `VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT` bit of the `flags` member, `handleTypes` member of `VkExternalMemoryImageCreateInfo`, and the `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT` of the `usage` member in the `VkImageCreateInfo` structure passed to `vkCreateImage`.
- For images created with a depth/stencil format, the `memoryTypeBits` member is identical for all `VkImage` objects created with the same combination of values for the `format` member, the `tiling` member, the `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` bit and `VK_IMAGE_CREATE_PROTECTED_BIT` bit of the `flags` member, the `VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT` bit of the `flags` member, `handleTypes` member of `VkExternalMemoryImageCreateInfo`, and the `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT` of the `usage` member in the `VkImageCreateInfo` structure passed to `vkCreateImage`.
- If the memory requirements are for a `VkImage`, the `memoryTypeBits` member **must** not refer to a `VkMemoryType` with a `propertyFlags` that has the `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT` bit set if the `image` did not have `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT` bit set in the `usage` member of the `VkImageCreateInfo` structure passed to `vkCreateImage`.
- If the memory requirements are for a `VkBuffer`, the `memoryTypeBits` member **must** not refer to a `VkMemoryType` with a `propertyFlags` that has the `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT` bit set.

NOTE

The implication of this requirement is that lazily allocated memory is disallowed for buffers in all cases.

- The `size` member is identical for all `VkBuffer` objects created with the same combination of creation parameters specified in `VkBufferCreateInfo` and its `pNext` chain.
- The `size` member is identical for all `VkImage` objects created with the same combination of creation parameters specified in `VkImageCreateInfo` and its `pNext` chain.

NOTE

This, however, does not imply that they interpret the contents of the bound memory identically with each other. That additional guarantee, however, **can** be explicitly requested using `VK_IMAGE_CREATE_ALIAS_BIT`.

To determine the memory requirements for a buffer resource, call:

```
// Provided by VK_VERSION_1_1
void vkGetBufferMemoryRequirements2(
    VkDevice                device,
    const VkBufferMemoryRequirementsInfo2* pInfo,
    VkMemoryRequirements2*   pMemoryRequirements);
```

- **device** is the logical device that owns the buffer.
- **pInfo** is a pointer to a [VkBufferMemoryRequirementsInfo2](#) structure containing parameters required for the memory requirements query.
- **pMemoryRequirements** is a pointer to a [VkMemoryRequirements2](#) structure in which the memory requirements of the buffer object are returned.

Valid Usage (Implicit)

- VUID-vkGetBufferMemoryRequirements2-device-parameter **device** **must** be a valid [VkDevice](#) handle
- VUID-vkGetBufferMemoryRequirements2-pInfo-parameter **pInfo** **must** be a valid pointer to a valid [VkBufferMemoryRequirementsInfo2](#) structure
- VUID-vkGetBufferMemoryRequirements2-pMemoryRequirements-parameter **pMemoryRequirements** **must** be a valid pointer to a [VkMemoryRequirements2](#) structure

The [VkBufferMemoryRequirementsInfo2](#) structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkBufferMemoryRequirementsInfo2 {
    VkStructureType    sType;
    const void*        pNext;
    VkBuffer           buffer;
} VkBufferMemoryRequirementsInfo2;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is `NULL` or a pointer to a structure extending this structure.
- **buffer** is the buffer to query.

Valid Usage (Implicit)

- VUID-VkBufferMemoryRequirementsInfo2-sType-sType **sType** **must** be `VK_STRUCTURE_TYPE_BUFFER_MEMORY_REQUIREMENTS_INFO_2`
- VUID-VkBufferMemoryRequirementsInfo2-pNext-pNext **pNext** **must** be `NULL`
- VUID-VkBufferMemoryRequirementsInfo2-buffer-parameter **buffer** **must** be a valid [VkBuffer](#) handle

To determine the memory requirements for an image resource, call:


```
// Provided by VK_VERSION_1_1
void vkGetImageMemoryRequirements2(
    VkDevice                device,
    const VkImageMemoryRequirementsInfo2* pInfo,
    VkMemoryRequirements2* pMemoryRequirements);
```

- **device** is the logical device that owns the image.
- **pInfo** is a pointer to a [VkImageMemoryRequirementsInfo2](#) structure containing parameters required for the memory requirements query.
- **pMemoryRequirements** is a pointer to a [VkMemoryRequirements2](#) structure in which the memory requirements of the image object are returned.

Valid Usage (Implicit)

- VUID-vkGetImageMemoryRequirements2-device-parameter **device** **must** be a valid [VkDevice](#) handle
- VUID-vkGetImageMemoryRequirements2-pInfo-parameter **pInfo** **must** be a valid pointer to a valid [VkImageMemoryRequirementsInfo2](#) structure
- VUID-vkGetImageMemoryRequirements2-pMemoryRequirements-parameter **pMemoryRequirements** **must** be a valid pointer to a [VkMemoryRequirements2](#) structure

The [VkImageMemoryRequirementsInfo2](#) structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkImageMemoryRequirementsInfo2 {
    VkStructureType    sType;
    const void*        pNext;
    VkImage            image;
} VkImageMemoryRequirementsInfo2;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is `NULL` or a pointer to a structure extending this structure.
- **image** is the image to query.

Valid Usage

- VUID-VkImageMemoryRequirementsInfo2-image-01589
If **image** was created with a *multi-planar* format and the `VK_IMAGE_CREATE_DISJOINT_BIT` flag, there **must** be a [VkImagePlaneMemoryRequirementsInfo](#) included in the **pNext** chain of the [VkImageMemoryRequirementsInfo2](#) structure
- VUID-VkImageMemoryRequirementsInfo2-image-02279
If **image** was created with `VK_IMAGE_CREATE_DISJOINT_BIT` and with

`VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then there **must** be a `VkImagePlaneMemoryRequirementsInfo` included in the `pNext` chain of the `VkImageMemoryRequirementsInfo2` structure

- VUID-VkImageMemoryRequirementsInfo2-image-01590

If `image` was not created with the `VK_IMAGE_CREATE_DISJOINT_BIT` flag, there **must** not be a `VkImagePlaneMemoryRequirementsInfo` included in the `pNext` chain of the `VkImageMemoryRequirementsInfo2` structure

- VUID-VkImageMemoryRequirementsInfo2-image-02280

If `image` was created with a single-plane format and with any `tiling` other than `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then there **must** not be a `VkImagePlaneMemoryRequirementsInfo` included in the `pNext` chain of the `VkImageMemoryRequirementsInfo2` structure

- VUID-VkImageMemoryRequirementsInfo2-image-08961

If `image` was created with the `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX` external memory handle type, then `image` **must** be bound to memory

Valid Usage (Implicit)

- VUID-VkImageMemoryRequirementsInfo2-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_MEMORY_REQUIREMENTS_INFO_2`

- VUID-VkImageMemoryRequirementsInfo2-pNext-pNext

`pNext` **must** be `NULL` or a pointer to a valid instance of `VkImagePlaneMemoryRequirementsInfo`

- VUID-VkImageMemoryRequirementsInfo2-sType-unique

The `sType` value of each structure in the `pNext` chain **must** be unique

- VUID-VkImageMemoryRequirementsInfo2-image-parameter

`image` **must** be a valid `VkImage` handle

To determine the memory requirements for a plane of a disjoint image, add a `VkImagePlaneMemoryRequirementsInfo` structure to the `pNext` chain of the `VkImageMemoryRequirementsInfo2` structure.

The `VkImagePlaneMemoryRequirementsInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkImagePlaneMemoryRequirementsInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkImageAspectFlags planeAspect;
} VkImagePlaneMemoryRequirementsInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- `planeAspect` is a `VkImageAspectFlagBits` value specifying the aspect corresponding to the image plane to query.

Valid Usage

- VUID-VkImagePlaneMemoryRequirementsInfo-planeAspect-02281
If the image's `tiling` is `VK_IMAGE_TILING_LINEAR` or `VK_IMAGE_TILING_OPTIMAL`, then `planeAspect` **must** be a single valid `multi-planar aspect mask` bit
- VUID-VkImagePlaneMemoryRequirementsInfo-planeAspect-02282
If the image's `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `planeAspect` **must** be a single valid *memory plane* for the image (that is, `aspectMask` **must** specify a plane index that is less than the `VkDrmFormatModifierPropertiesEXT::drmFormatModifierPlaneCount` associated with the image's `format` and `VkImageDrmFormatModifierPropertiesEXT::drmFormatModifier`)

Valid Usage (Implicit)

- VUID-VkImagePlaneMemoryRequirementsInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_PLANE_MEMORY_REQUIREMENTS_INFO`
- VUID-VkImagePlaneMemoryRequirementsInfo-planeAspect-parameter
`planeAspect` **must** be a valid `VkImageAspectFlagBits` value

The `VkMemoryRequirements2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkMemoryRequirements2 {
    VkStructureType    sType;
    void*              pNext;
    VkMemoryRequirements memoryRequirements;
} VkMemoryRequirements2;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `memoryRequirements` is a `VkMemoryRequirements` structure describing the memory requirements of the resource.

Valid Usage (Implicit)

- VUID-VkMemoryRequirements2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_REQUIREMENTS_2`
- VUID-VkMemoryRequirements2-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkMemoryDedicatedRequirements`
- VUID-VkMemoryRequirements2-sType-unique

The `sType` value of each structure in the `pNext` chain **must** be unique

The `VkMemoryDedicatedRequirements` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkMemoryDedicatedRequirements {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           prefersDedicatedAllocation;
    VkBool32           requiresDedicatedAllocation;
} VkMemoryDedicatedRequirements;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `prefersDedicatedAllocation` specifies that the implementation would prefer a dedicated allocation for this resource. The application is still free to suballocate the resource but it **may** get better performance if a dedicated allocation is used.
- `requiresDedicatedAllocation` specifies that a dedicated allocation is required for this resource.

To determine the dedicated allocation requirements of a buffer or image resource, add a `VkMemoryDedicatedRequirements` structure to the `pNext` chain of the `VkMemoryRequirements2` structure passed as the `pMemoryRequirements` parameter of `vkGetBufferMemoryRequirements2`, or `vkGetImageMemoryRequirements2` respectively.

Constraints on the values returned for buffer resources are:

- `requiresDedicatedAllocation` **may** be `VK_TRUE` if the `pNext` chain of `VkBufferCreateInfo` for the call to `vkCreateBuffer` used to create the buffer being queried included a `VkExternalMemoryBufferCreateInfo` structure, and any of the handle types specified in `VkExternalMemoryBufferCreateInfo::handleTypes` requires dedicated allocation, as reported by `vkGetPhysicalDeviceExternalBufferProperties` in `VkExternalBufferProperties::externalMemoryProperties.externalMemoryFeatures`. Otherwise, `requiresDedicatedAllocation` will be `VK_FALSE`.
- When the implementation sets `requiresDedicatedAllocation` to `VK_TRUE`, it **must** also set `prefersDedicatedAllocation` to `VK_TRUE`.
- If `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` was set in `VkBufferCreateInfo::flags` when `buffer` was created, then both `prefersDedicatedAllocation` and `requiresDedicatedAllocation` will be `VK_FALSE`.

Constraints on the values returned for image resources are:

- `requiresDedicatedAllocation` **may** be `VK_TRUE` if the `pNext` chain of `VkImageCreateInfo` for the call to `vkCreateImage` used to create the image being queried included a `VkExternalMemoryImageCreateInfo` structure, and any of the handle types specified in `VkExternalMemoryImageCreateInfo::handleTypes` requires dedicated allocation, as reported by `vkGetPhysicalDeviceImageFormatProperties2` in `VkExternalImageFormatProperties::externalMemoryProperties.externalMemoryFeatures`.

- `requiresDedicatedAllocation` may be `VK_TRUE` if the image's tiling is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`.
- `requiresDedicatedAllocation` will otherwise be `VK_FALSE`
- If `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` was set in `VkImageCreateInfo::flags` when `image` was created, then both `prefersDedicatedAllocation` and `requiresDedicatedAllocation` will be `VK_FALSE`.

Valid Usage (Implicit)

- VUID-VkMemoryDedicatedRequirements-sType-sType
`sType` must be `VK_STRUCTURE_TYPE_MEMORY_DEDICATED_REQUIREMENTS`

To attach memory to a buffer object, call:

```
// Provided by VK_VERSION_1_0
VkResult vkBindBufferMemory(
    VkDevice          device,
    VkBuffer          buffer,
    VkDeviceMemory    memory,
    VkDeviceSize      memoryOffset);
```

- `device` is the logical device that owns the buffer and memory.
- `buffer` is the buffer to be attached to memory.
- `memory` is a `VkDeviceMemory` object describing the device memory to attach.
- `memoryOffset` is the start offset of the region of `memory` which is to be bound to the buffer. The number of bytes returned in the `VkMemoryRequirements::size` member in `memory`, starting from `memoryOffset` bytes, will be bound to the specified buffer.

`vkBindBufferMemory` is equivalent to passing the same parameters through `VkBindBufferMemoryInfo` to `vkBindBufferMemory2`.

If the `memory` was obtained by a memory import operation with `VkExternalMemoryBufferCreateInfo::handleTypes` assigned to `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV`, the properties of `buffer` and the `memoryOffset` must be compatible with the attributes used to create `NvSciBufObj`, otherwise the implementation will return `VK_ERROR_VALIDATION_FAILED`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkBindBufferMemory` must not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkBindBufferMemory-buffer-07459
`buffer` must not have been bound to a memory object
- VUID-vkBindBufferMemory-buffer-01030

buffer **must** not have been created with any sparse memory binding flags

- VUID-vkBindBufferMemory-memoryOffset-01031
memoryOffset **must** be less than the size of **memory**
- VUID-vkBindBufferMemory-memory-01035
memory **must** have been allocated using one of the memory types allowed in the **memoryTypeBits** member of the **VkMemoryRequirements** structure returned from a call to **vkGetBufferMemoryRequirements** with **buffer**
- VUID-vkBindBufferMemory-None-10739
memoryOffset **must** be an integer multiple of the **alignment** member of the **VkMemoryRequirements** structure returned from a call to **vkGetBufferMemoryRequirements** with **buffer**
- VUID-vkBindBufferMemory-None-10741
size member of the **VkMemoryRequirements** structure returned from a call to **vkGetBufferMemoryRequirements** with **buffer** **must** be less than or equal to the size of **memory** minus **memoryOffset**
- VUID-vkBindBufferMemory-buffer-01444
If **buffer** requires a dedicated allocation (as reported by **vkGetBufferMemoryRequirements2** in **VkMemoryDedicatedRequirements::requiresDedicatedAllocation** for **buffer**), **memory** **must** have been allocated with **VkMemoryDedicatedAllocateInfo::buffer** equal to **buffer**
- VUID-vkBindBufferMemory-memory-01508
If the **VkMemoryAllocateInfo** provided when **memory** was allocated included a **VkMemoryDedicatedAllocateInfo** structure in its **pNext** chain, and **VkMemoryDedicatedAllocateInfo::buffer** was not **VK_NULL_HANDLE**, then **buffer** **must** equal **VkMemoryDedicatedAllocateInfo::buffer**, and **memoryOffset** **must** be zero
- VUID-vkBindBufferMemory-memory-10925
If the **VkMemoryAllocateInfo** provided when **memory** was allocated included a **VkMemoryDedicatedAllocateInfo** structure in its **pNext** chain, **VkMemoryDedicatedAllocateInfo::image** **must** have been **VK_NULL_HANDLE**
- VUID-vkBindBufferMemory-None-01898
If **buffer** was created with the **VK_BUFFER_CREATE_PROTECTED_BIT** bit set, the **buffer** **must** be bound to a memory object allocated with a memory type that reports **VK_MEMORY_PROPERTY_PROTECTED_BIT**
- VUID-vkBindBufferMemory-None-01899
If **buffer** was created with the **VK_BUFFER_CREATE_PROTECTED_BIT** bit not set, the **buffer** **must** not be bound to a memory object allocated with a memory type that reports **VK_MEMORY_PROPERTY_PROTECTED_BIT**
- VUID-vkBindBufferMemory-memory-02726
If the value of **VkExportMemoryAllocateInfo::handleTypes** used to allocate **memory** is not **0**, it **must** include at least one of the handles set in **VkExternalMemoryBufferCreateInfo::handleTypes** when **buffer** was created
- VUID-vkBindBufferMemory-memory-02985
If **memory** was allocated by a memory import operation, the external handle type of the

imported memory **must** also have been set in `VkExternalMemoryBufferCreateInfo::handleTypes` when `buffer` was created

- VUID-vkBindBufferMemory-bufferDeviceAddress-03339

If the `VkPhysicalDeviceBufferDeviceAddressFeatures::bufferDeviceAddress` feature is enabled and `buffer` was created with the `VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT` bit set, memory **must** have been allocated with the `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT` bit set

- VUID-vkBindBufferMemory-bufferDeviceAddressCaptureReplay-09200

If the `VkPhysicalDeviceBufferDeviceAddressFeatures::bufferDeviceAddressCaptureReplay` feature is enabled and `buffer` was created with the `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT` bit set, memory **must** have been allocated with the `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT` bit set

Valid Usage (Implicit)

- VUID-vkBindBufferMemory-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkBindBufferMemory-buffer-parameter
`buffer` **must** be a valid `VkBuffer` handle
- VUID-vkBindBufferMemory-memory-parameter
`memory` **must** be a valid `VkDeviceMemory` handle
- VUID-vkBindBufferMemory-buffer-parent
`buffer` **must** have been created, allocated, or retrieved from `device`
- VUID-vkBindBufferMemory-memory-parent
`memory` **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `buffer` **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To attach memory to buffer objects for one or more buffers at a time, call:

```
// Provided by VK_VERSION_1_1
VkResult vkBindBufferMemory2(
    VkDevice          device,
    uint32_t          bindInfoCount,
    const VkBindBufferMemoryInfo* pBindInfos);
```

- **device** is the logical device that owns the buffers and memory.
- **bindInfoCount** is the number of elements in **pBindInfos**.
- **pBindInfos** is a pointer to an array of **bindInfoCount** **VkBindBufferMemoryInfo** structures describing buffers and memory to bind.

On some implementations, it **may** be more efficient to batch memory bindings into a single command.

If any of the memory binding operations described by **pBindInfos** fail, the **VkResult** returned by this command **must** be the return value of any one of the memory binding operations which did not return **VK_SUCCESS**.

NOTE

If the **vkBindBufferMemory2** command failed, and **bindInfoCount** was greater than one, then the buffers referenced by **pBindInfos** will be in an indeterminate state, and must not be used.

Applications should destroy these buffers.

If **VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations** is **VK_TRUE**, **vkBindBufferMemory2** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage (Implicit)

- VUID-vkBindBufferMemory2-device-parameter
device **must** be a valid **VkDevice** handle
- VUID-vkBindBufferMemory2-pBindInfos-parameter
pBindInfos **must** be a valid pointer to an array of **bindInfoCount** valid **VkBindBufferMemoryInfo** structures
- VUID-vkBindBufferMemory2-bindInfoCount-arraylength
bindInfoCount **must** be greater than 0

Return Codes

Success

- **VK_SUCCESS**

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

`VkBindBufferMemoryInfo` contains members corresponding to the parameters of `vkBindBufferMemory`.

The `VkBindBufferMemoryInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkBindBufferMemoryInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkBuffer            buffer;
    VkDeviceMemory      memory;
    VkDeviceSize        memoryOffset;
} VkBindBufferMemoryInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `buffer` is the buffer to be attached to memory.
- `memory` is a `VkDeviceMemory` object describing the device memory to attach.
- `memoryOffset` is the start offset of the region of `memory` which is to be bound to the buffer. The number of bytes returned in the `VkMemoryRequirements::size` member in `memory`, starting from `memoryOffset` bytes, will be bound to the specified buffer.

Valid Usage

- VUID-VkBindBufferMemoryInfo-buffer-07459
`buffer` **must** not have been bound to a memory object
- VUID-VkBindBufferMemoryInfo-buffer-01030
`buffer` **must** not have been created with any sparse memory binding flags
- VUID-VkBindBufferMemoryInfo-memoryOffset-01031
`memoryOffset` **must** be less than the size of `memory`
- VUID-VkBindBufferMemoryInfo-memory-01035
`memory` **must** have been allocated using one of the memory types allowed in the `memoryTypeBits` member of the `VkMemoryRequirements` structure returned from a call to `vkGetBufferMemoryRequirements` with `buffer`
- VUID-VkBindBufferMemoryInfo-None-10739
`memoryOffset` **must** be an integer multiple of the `alignment` member of the

`VkMemoryRequirements` structure returned from a call to `vkGetBufferMemoryRequirements` with `buffer`

- VUID-VkBindBufferMemoryInfo-None-10741
`size` member of the `VkMemoryRequirements` structure returned from a call to `vkGetBufferMemoryRequirements` with `buffer` **must** be less than or equal to the size of `memory` minus `memoryOffset`
- VUID-VkBindBufferMemoryInfo-buffer-01444
If `buffer` requires a dedicated allocation (as reported by `vkGetBufferMemoryRequirements2` in `VkMemoryDedicatedRequirements::requiresDedicatedAllocation` for `buffer`), `memory` **must** have been allocated with `VkMemoryDedicatedAllocateInfo::buffer` equal to `buffer`
- VUID-VkBindBufferMemoryInfo-memory-01508
If the `VkMemoryAllocateInfo` provided when `memory` was allocated included a `VkMemoryDedicatedAllocateInfo` structure in its `pNext` chain, and `VkMemoryDedicatedAllocateInfo::buffer` was not `VK_NULL_HANDLE`, then `buffer` **must** equal `VkMemoryDedicatedAllocateInfo::buffer`, and `memoryOffset` **must** be zero
- VUID-VkBindBufferMemoryInfo-memory-10925
If the `VkMemoryAllocateInfo` provided when `memory` was allocated included a `VkMemoryDedicatedAllocateInfo` structure in its `pNext` chain, `VkMemoryDedicatedAllocateInfo::image` **must** have been `VK_NULL_HANDLE`
- VUID-VkBindBufferMemoryInfo-None-01898
If `buffer` was created with the `VK_BUFFER_CREATE_PROTECTED_BIT` bit set, the buffer **must** be bound to a memory object allocated with a memory type that reports `VK_MEMORY_PROPERTY_PROTECTED_BIT`
- VUID-VkBindBufferMemoryInfo-None-01899
If `buffer` was created with the `VK_BUFFER_CREATE_PROTECTED_BIT` bit not set, the buffer **must** not be bound to a memory object allocated with a memory type that reports `VK_MEMORY_PROPERTY_PROTECTED_BIT`
- VUID-VkBindBufferMemoryInfo-memory-02726
If the value of `VkExportMemoryAllocateInfo::handleTypes` used to allocate `memory` is not `0`, it **must** include at least one of the handles set in `VkExternalMemoryBufferCreateInfo::handleTypes` when `buffer` was created
- VUID-VkBindBufferMemoryInfo-memory-02985
If `memory` was allocated by a memory import operation, the external handle type of the imported memory **must** also have been set in `VkExternalMemoryBufferCreateInfo::handleTypes` when `buffer` was created
- VUID-VkBindBufferMemoryInfo-bufferDeviceAddress-03339
If the `VkPhysicalDeviceBufferDeviceAddressFeatures::bufferDeviceAddress` feature is enabled and `buffer` was created with the `VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT` bit set, `memory` **must** have been allocated with the `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT` bit set
- VUID-VkBindBufferMemoryInfo-bufferDeviceAddressCaptureReplay-09200
If the `VkPhysicalDeviceBufferDeviceAddressFeatures::bufferDeviceAddressCaptureReplay` feature is enabled and `buffer` was created with the

`VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT` bit set, `memory` **must** have been allocated with the `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT` bit set

- VUID-VkBindBufferMemoryInfo-pNext-01605

If the `pNext` chain includes a `VkBindBufferMemoryDeviceGroupInfo` structure, all instances of `memory` specified by `VkBindBufferMemoryDeviceGroupInfo::pDeviceIndices` **must** have been allocated

Valid Usage (Implicit)

- VUID-VkBindBufferMemoryInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_INFO`
- VUID-VkBindBufferMemoryInfo-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkBindBufferMemoryDeviceGroupInfo`
- VUID-VkBindBufferMemoryInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkBindBufferMemoryInfo-buffer-parameter
`buffer` **must** be a valid `VkBuffer` handle
- VUID-VkBindBufferMemoryInfo-memory-parameter
`memory` **must** be a valid `VkDeviceMemory` handle
- VUID-VkBindBufferMemoryInfo-commonparent
Both of `buffer`, and `memory` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `buffer` **must** be externally synchronized

The `VkBindBufferMemoryDeviceGroupInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkBindBufferMemoryDeviceGroupInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           deviceIndexCount;
    const uint32_t*     pDeviceIndices;
} VkBindBufferMemoryDeviceGroupInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `deviceIndexCount` is the number of elements in `pDeviceIndices`.

- `pDeviceIndices` is a pointer to an array of device indices.

If the `pNext` chain of `VkBindBufferMemoryInfo` includes a `VkBindBufferMemoryDeviceGroupInfo` structure, then that structure determines how memory is bound to buffers across multiple devices in a device group.

If `deviceIndexCount` is greater than zero, then on device index `i` the buffer is attached to the instance of `memory` on the physical device with device index `pDeviceIndices[i]`.

If `deviceIndexCount` is zero and `memory` comes from a memory heap with the `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` bit set, then it is as if `pDeviceIndices` contains consecutive indices from zero to the number of physical devices in the logical device, minus one. In other words, by default each physical device attaches to its own instance of `memory`.

If `deviceIndexCount` is zero and `memory` comes from a memory heap without the `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` bit set, then it is as if `pDeviceIndices` contains an array of zeros. In other words, by default each physical device attaches to instance zero.

Valid Usage

- VUID-VkBindBufferMemoryDeviceGroupInfo-deviceIndexCount-01606
`deviceIndexCount` **must** either be zero or equal to the number of physical devices in the logical device
- VUID-VkBindBufferMemoryDeviceGroupInfo-pDeviceIndices-01607
All elements of `pDeviceIndices` **must** be valid device indices

Valid Usage (Implicit)

- VUID-VkBindBufferMemoryDeviceGroupInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_DEVICE_GROUP_INFO`
- VUID-VkBindBufferMemoryDeviceGroupInfo-pDeviceIndices-parameter
If `deviceIndexCount` is not 0, `pDeviceIndices` **must** be a valid pointer to an array of `deviceIndexCount` `uint32_t` values

To attach memory to a `VkImage` object created without the `VK_IMAGE_CREATE_DISJOINT_BIT` set, call:

```
// Provided by VK_VERSION_1_0
VkResult vkBindImageMemory(
    VkDevice          device,
    VkImage           image,
    VkDeviceMemory    memory,
    VkDeviceSize      memoryOffset);
```

- `device` is the logical device that owns the image and memory.
- `image` is the image.

- `memory` is the `VkDeviceMemory` object describing the device memory to attach.
- `memoryOffset` is the start offset of the region of `memory` which is to be bound to the image. The number of bytes returned in the `VkMemoryRequirements::size` member in `memory`, starting from `memoryOffset` bytes, will be bound to the specified image.

`vkBindImageMemory` is equivalent to passing the same parameters through `VkBindImageMemoryInfo` to `vkBindImageMemory2`.

If the `memory` is allocated by a memory import operation with `VkExternalMemoryBufferCreateInfo::handleTypes` assigned to `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV`, the properties of `image` and the `memoryOffset` **must** be compatible with the attributes used to create `NvSciBufObj`, otherwise the implementation will return `VK_ERROR_VALIDATION_FAILED`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkBindImageMemory` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkBindImageMemory-image-07460
`image` **must** not have been bound to a memory object
- VUID-vkBindImageMemory-image-01045
`image` **must** not have been created with any sparse memory binding flags
- VUID-vkBindImageMemory-memoryOffset-01046
`memoryOffset` **must** be less than the size of `memory`
- VUID-vkBindImageMemory-image-01445
If `image` requires a dedicated allocation (as reported by `vkGetImageMemoryRequirements2` in `VkMemoryDedicatedRequirements::requiresDedicatedAllocation` for `image`), `memory` **must** have been created with `VkMemoryDedicatedAllocateInfo::image` equal to `image`
- VUID-vkBindImageMemory-memory-02628
If the `VkMemoryAllocateInfo` provided when `memory` was allocated included a `VkMemoryDedicatedAllocateInfo` structure in its `pNext` chain, and `VkMemoryDedicatedAllocateInfo::image` was not `VK_NULL_HANDLE`, then `image` **must** equal `VkMemoryDedicatedAllocateInfo::image` and `memoryOffset` **must** be zero
- VUID-vkBindImageMemory-memory-10926
If the `VkMemoryAllocateInfo` provided when `memory` was allocated included a `VkMemoryDedicatedAllocateInfo` structure in its `pNext` chain, `VkMemoryDedicatedAllocateInfo::buffer` **must** have been `VK_NULL_HANDLE`
- VUID-vkBindImageMemory-None-01901
If `image` was created with the `VK_IMAGE_CREATE_PROTECTED_BIT` bit set, the `image` **must** be bound to a memory object allocated with a memory type that reports `VK_MEMORY_PROPERTY_PROTECTED_BIT`
- VUID-vkBindImageMemory-None-01902
If `image` was created with the `VK_IMAGE_CREATE_PROTECTED_BIT` bit not set, the `image` **must** not be bound to a memory object created with a memory type that reports

VK_MEMORY_PROPERTY_PROTECTED_BIT

- VUID-vkBindImageMemory-memory-02728
If the value of `VkExportMemoryAllocateInfo::handleTypes` used to allocate `memory` is not `0`, it **must** include at least one of the handles set in `VkExternalMemoryImageCreateInfo::handleTypes` when `image` was created
- VUID-vkBindImageMemory-memory-02989
If `memory` was created by a memory import operation, the external handle type of the imported memory **must** also have been set in `VkExternalMemoryImageCreateInfo::handleTypes` when `image` was created
- VUID-vkBindImageMemory-image-01608
`image` **must** not have been created with the `VK_IMAGE_CREATE_DISJOINT_BIT` set
- VUID-vkBindImageMemory-memory-01047
`memory` **must** have been allocated using one of the memory types allowed in the `memoryTypeBits` member of the `VkMemoryRequirements` structure returned from a call to `vkGetImageMemoryRequirements` with `image`
- VUID-vkBindImageMemory-None-10735
`memoryOffset` **must** be an integer multiple of the `alignment` member of the `VkMemoryRequirements` structure returned from a call to `vkGetImageMemoryRequirements` with `image`
- VUID-vkBindImageMemory-None-10737
`size` member of the `VkMemoryRequirements` structure returned from a call to `vkGetImageMemoryRequirements` with `image` **must** be less than or equal to the size of `memory` minus `memoryOffset`

Valid Usage (Implicit)

- VUID-vkBindImageMemory-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkBindImageMemory-image-parameter
`image` **must** be a valid `VkImage` handle
- VUID-vkBindImageMemory-memory-parameter
`memory` **must** be a valid `VkDeviceMemory` handle
- VUID-vkBindImageMemory-image-parent
`image` **must** have been created, allocated, or retrieved from `device`
- VUID-vkBindImageMemory-memory-parent
`memory` **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `image` **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To attach memory to image objects for one or more images at a time, call:

```
// Provided by VK_VERSION_1_1
VkResult vkBindImageMemory2(
    VkDevice          device,
    uint32_t          bindInfoCount,
    const VkBindImageMemoryInfo* pBindInfos);
```

- `device` is the logical device that owns the images and memory.
- `bindInfoCount` is the number of elements in `pBindInfos`.
- `pBindInfos` is a pointer to an array of `VkBindImageMemoryInfo` structures, describing images and memory to bind.

On some implementations, it **may** be more efficient to batch memory bindings into a single command.

If any of the memory binding operations described by `pBindInfos` fail, the `VkResult` returned by this command **must** be the return value of any one of the memory binding operations which did not return `VK_SUCCESS`.

NOTE

If the `vkBindImageMemory2` command failed, and `bindInfoCount` was greater than one, then the images referenced by `pBindInfos` will be in an indeterminate state, and must not be used.

Applications should destroy these images.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkBindImageMemory2` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkBindImageMemory2-pBindInfos-02858

If any `VkBindImageMemoryInfo::image` was created with `VK_IMAGE_CREATE_DISJOINT_BIT` then all planes of `VkBindImageMemoryInfo::image` **must** be bound individually in

separate `pBindInfos`

- VUID-vkBindImageMemory2-pBindInfos-04006
`pBindInfos` **must** not refer to the same image subresource more than once

Valid Usage (Implicit)

- VUID-vkBindImageMemory2-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkBindImageMemory2-pBindInfos-parameter
`pBindInfos` **must** be a valid pointer to an array of `bindInfoCount` valid `VkBindImageMemoryInfo` structures
- VUID-vkBindImageMemory2-bindInfoCount-arraylength
`bindInfoCount` **must** be greater than 0

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

`VkBindImageMemoryInfo` contains members corresponding to the parameters of `vkBindImageMemory`.

The `VkBindImageMemoryInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkBindImageMemoryInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkImage             image;
    VkDeviceMemory      memory;
    VkDeviceSize        memoryOffset;
} VkBindImageMemoryInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `image` is the image to be attached to memory.

- `memory` is a `VkDeviceMemory` object describing the device memory to attach.
- `memoryOffset` is the start offset of the region of `memory` which is to be bound to the image. The number of bytes returned in the `VkMemoryRequirements::size` member in `memory`, starting from `memoryOffset` bytes, will be bound to the specified image.

Valid Usage

- VUID-VkBindImageMemoryInfo-image-07460
`image` **must** not have been bound to a memory object
- VUID-VkBindImageMemoryInfo-image-01045
`image` **must** not have been created with any sparse memory binding flags
- VUID-VkBindImageMemoryInfo-memoryOffset-01046
`memoryOffset` **must** be less than the size of `memory`
- VUID-VkBindImageMemoryInfo-image-01445
If `image` requires a dedicated allocation (as reported by `vkGetImageMemoryRequirements2` in `VkMemoryDedicatedRequirements::requiresDedicatedAllocation` for `image`), `memory` **must** have been created with `VkMemoryDedicatedAllocateInfo::image` equal to `image`
- VUID-VkBindImageMemoryInfo-memory-02628
If the `VkMemoryAllocateInfo` provided when `memory` was allocated included a `VkMemoryDedicatedAllocateInfo` structure in its `pNext` chain, and `VkMemoryDedicatedAllocateInfo::image` was not `VK_NULL_HANDLE`, then `image` **must** equal `VkMemoryDedicatedAllocateInfo::image` and `memoryOffset` **must** be zero
- VUID-VkBindImageMemoryInfo-memory-10926
If the `VkMemoryAllocateInfo` provided when `memory` was allocated included a `VkMemoryDedicatedAllocateInfo` structure in its `pNext` chain, `VkMemoryDedicatedAllocateInfo::buffer` **must** have been `VK_NULL_HANDLE`
- VUID-VkBindImageMemoryInfo-None-01901
If image was created with the `VK_IMAGE_CREATE_PROTECTED_BIT` bit set, the image **must** be bound to a memory object allocated with a memory type that reports `VK_MEMORY_PROPERTY_PROTECTED_BIT`
- VUID-VkBindImageMemoryInfo-None-01902
If image was created with the `VK_IMAGE_CREATE_PROTECTED_BIT` bit not set, the image **must** not be bound to a memory object created with a memory type that reports `VK_MEMORY_PROPERTY_PROTECTED_BIT`
- VUID-VkBindImageMemoryInfo-memory-02728
If the value of `VkExportMemoryAllocateInfo::handleTypes` used to allocate `memory` is not `0`, it **must** include at least one of the handles set in `VkExternalMemoryImageCreateInfo::handleTypes` when `image` was created
- VUID-VkBindImageMemoryInfo-memory-02989
If `memory` was created by a memory import operation, the external handle type of the imported memory **must** also have been set in `VkExternalMemoryImageCreateInfo::handleTypes` when `image` was created

- VUID-VkBindImageMemoryInfo-pNext-01615
If the `pNext` chain does not include a `VkBindImagePlaneMemoryInfo` structure, `memory` **must** have been allocated using one of the memory types allowed in the `memoryTypeBits` member of the `VkMemoryRequirements` structure returned from a call to `vkGetImageMemoryRequirements2` with `image`
- VUID-VkBindImageMemoryInfo-pNext-01616
If the `pNext` chain does not include a `VkBindImagePlaneMemoryInfo` structure, `memoryOffset` **must** be an integer multiple of the `alignment` member of the `VkMemoryRequirements` structure returned from a call to `vkGetImageMemoryRequirements2` with `image`
- VUID-VkBindImageMemoryInfo-pNext-01617
If the `pNext` chain does not include a `VkBindImagePlaneMemoryInfo` structure, the difference of the size of `memory` and `memoryOffset` **must** be greater than or equal to the `size` member of the `VkMemoryRequirements` structure returned from a call to `vkGetImageMemoryRequirements2` with the same `image`
- VUID-VkBindImageMemoryInfo-pNext-01618
If the `pNext` chain includes a `VkBindImagePlaneMemoryInfo` structure, `image` **must** have been created with the `VK_IMAGE_CREATE_DISJOINT_BIT` bit set
- VUID-VkBindImageMemoryInfo-image-07736
If `image` was created with the `VK_IMAGE_CREATE_DISJOINT_BIT` bit set, then the `pNext` chain **must** include a `VkBindImagePlaneMemoryInfo` structure
- VUID-VkBindImageMemoryInfo-pNext-01619
If the `pNext` chain includes a `VkBindImagePlaneMemoryInfo` structure, `memory` **must** have been allocated using one of the memory types allowed in the `memoryTypeBits` member of the `VkMemoryRequirements` structure returned from a call to `vkGetImageMemoryRequirements2` with `image` and where `VkBindImagePlaneMemoryInfo::planeAspect` corresponds to the `VkImagePlaneMemoryRequirementsInfo::planeAspect` in the `VkImageMemoryRequirementsInfo2` structure's `pNext` chain
- VUID-VkBindImageMemoryInfo-pNext-01620
If the `pNext` chain includes a `VkBindImagePlaneMemoryInfo` structure, `memoryOffset` **must** be an integer multiple of the `alignment` member of the `VkMemoryRequirements` structure returned from a call to `vkGetImageMemoryRequirements2` with `image` and where `VkBindImagePlaneMemoryInfo::planeAspect` corresponds to the `VkImagePlaneMemoryRequirementsInfo::planeAspect` in the `VkImageMemoryRequirementsInfo2` structure's `pNext` chain
- VUID-VkBindImageMemoryInfo-pNext-01621
If the `pNext` chain includes a `VkBindImagePlaneMemoryInfo` structure, the difference of the size of `memory` and `memoryOffset` **must** be greater than or equal to the `size` member of the `VkMemoryRequirements` structure returned from a call to `vkGetImageMemoryRequirements2` with the same `image` and where `VkBindImagePlaneMemoryInfo::planeAspect` corresponds to the `VkImagePlaneMemoryRequirementsInfo::planeAspect` in the `VkImageMemoryRequirementsInfo2` structure's `pNext` chain
- VUID-VkBindImageMemoryInfo-pNext-01626

If the **pNext** chain includes a **VkBindImageMemoryDeviceGroupInfo** structure, all instances of **memory** specified by **VkBindImageMemoryDeviceGroupInfo::pDeviceIndices** **must** have been allocated

- VUID-VkBindImageMemoryInfo-image-01630

If **image** was created with a valid swapchain handle in **VkImageSwapchainCreateInfoKHR::swapchain**, then the **pNext** chain **must** include a **VkBindImageMemorySwapchainInfoKHR** structure containing the same swapchain handle

- VUID-VkBindImageMemoryInfo-pNext-01631

If the **pNext** chain includes a **VkBindImageMemorySwapchainInfoKHR** structure, **memory** **must** be **VK_NULL_HANDLE**

- VUID-VkBindImageMemoryInfo-pNext-01632

If the **pNext** chain does not include a **VkBindImageMemorySwapchainInfoKHR** structure, **memory** **must** be a valid **VkDeviceMemory** handle

Valid Usage (Implicit)

- VUID-VkBindImageMemoryInfo-sType-sType

sType **must** be **VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_INFO**

- VUID-VkBindImageMemoryInfo-pNext-pNext

Each **pNext** member of any structure (including this one) in the **pNext** chain **must** be either **NULL** or a pointer to a valid instance of **VkBindImageMemoryDeviceGroupInfo**, **VkBindImageMemorySwapchainInfoKHR**, or **VkBindImagePlaneMemoryInfo**

- VUID-VkBindImageMemoryInfo-sType-unique

The **sType** value of each structure in the **pNext** chain **must** be unique

- VUID-VkBindImageMemoryInfo-image-parameter

image **must** be a valid **VkImage** handle

- VUID-VkBindImageMemoryInfo-commonparent

Both of **image**, and **memory** that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same **VkDevice**

Host Synchronization

- Host access to **image** **must** be externally synchronized

The **VkBindImageMemoryDeviceGroupInfo** structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkBindImageMemoryDeviceInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           deviceIndexCount;
    const uint32_t*     pDeviceIndices;
    uint32_t           splitInstanceBindRegionCount;
    const VkRect2D*     pSplitInstanceBindRegions;
} VkBindImageMemoryDeviceInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `deviceIndexCount` is the number of elements in `pDeviceIndices`.
- `pDeviceIndices` is a pointer to an array of device indices.
- `splitInstanceBindRegionCount` is the number of elements in `pSplitInstanceBindRegions`.
- `pSplitInstanceBindRegions` is a pointer to an array of `VkRect2D` structures describing which regions of the image are attached to each instance of memory.

If the `pNext` chain of `VkBindImageMemoryInfo` includes a `VkBindImageMemoryDeviceInfo` structure, then that structure determines how memory is bound to images across multiple devices in a device group.

If `deviceIndexCount` is greater than zero, then on device index `i` `image` is attached to the instance of the memory on the physical device with device index `pDeviceIndices[i]`.

In Vulkan SC, `splitInstanceBindRegionCount` **must** be zero because sparse allocations are not supported [SCID-8].

If `splitInstanceBindRegionCount` and `deviceIndexCount` are zero and the memory comes from a memory heap with the `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` bit set, then it is as if `pDeviceIndices` contains consecutive indices from zero to the number of physical devices in the logical device, minus one. In other words, by default each physical device attaches to its own instance of the memory.

If `splitInstanceBindRegionCount` and `deviceIndexCount` are zero and the memory comes from a memory heap without the `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT` bit set, then it is as if `pDeviceIndices` contains an array of zeros. In other words, by default each physical device attaches to instance zero.

Valid Usage

- VUID-VkBindImageMemoryDeviceInfo-deviceIndexCount-01634
`deviceIndexCount` **must** either be zero or equal to the number of physical devices in the logical device
- VUID-VkBindImageMemoryDeviceInfo-pDeviceIndices-01635
All elements of `pDeviceIndices` **must** be valid device indices

- VUID-VkBindImageMemoryDeviceGroupInfo-splitInstanceBindRegionCount-05067
`splitInstanceBindRegionCount` **must** be zero

Valid Usage (Implicit)

- VUID-VkBindImageMemoryDeviceGroupInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_DEVICE_GROUP_INFO`
- VUID-VkBindImageMemoryDeviceGroupInfo-pDeviceIndices-parameter
If `deviceIndexCount` is not 0, `pDeviceIndices` **must** be a valid pointer to an array of `deviceIndexCount` `uint32_t` values
- VUID-VkBindImageMemoryDeviceGroupInfo-pSplitInstanceBindRegions-parameter
If `splitInstanceBindRegionCount` is not 0, `pSplitInstanceBindRegions` **must** be a valid pointer to an array of `splitInstanceBindRegionCount` `VkRect2D` structures

If the `pNext` chain of `VkBindImageMemoryInfo` includes a `VkBindImageMemorySwapchainInfoKHR` structure, then that structure includes a swapchain handle and image index indicating that the image will be bound to memory from that swapchain.

The `VkBindImageMemorySwapchainInfoKHR` structure is defined as:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
typedef struct VkBindImageMemorySwapchainInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkSwapchainKHR     swapchain;
    uint32_t           imageIndex;
} VkBindImageMemorySwapchainInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `swapchain` is `VK_NULL_HANDLE` or a swapchain handle.
- `imageIndex` is an image index within `swapchain`.

If `swapchain` is not `NULL`, the `swapchain` and `imageIndex` are used to determine the memory that the image is bound to, instead of `memory` and `memoryOffset`.

Memory **can** be bound to a swapchain and use the `pDeviceIndices` or `pSplitInstanceBindRegions` members of `VkBindImageMemoryDeviceGroupInfo`.

Valid Usage

- VUID-VkBindImageMemorySwapchainInfoKHR-imageIndex-01644
`imageIndex` **must** be less than the number of images in `swapchain`

Valid Usage (Implicit)

- VUID-VkBindImageMemorySwapchainInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_SWAPCHAIN_INFO_KHR`
- VUID-VkBindImageMemorySwapchainInfoKHR-swapchain-parameter
`swapchain` **must** be a valid `VkSwapchainKHR` handle

Host Synchronization

- Host access to `swapchain` **must** be externally synchronized

In order to bind *planes* of a *disjoint image*, add a `VkBindImagePlaneMemoryInfo` structure to the `pNext` chain of `VkBindImageMemoryInfo`.

The `VkBindImagePlaneMemoryInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkBindImagePlaneMemoryInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkImageAspectFlagBits planeAspect;
} VkBindImagePlaneMemoryInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `planeAspect` is a `VkImageAspectFlagBits` value specifying the aspect of the disjoint image plane to bind.

Valid Usage

- VUID-VkBindImagePlaneMemoryInfo-planeAspect-02283
If the image's `tiling` is `VK_IMAGE_TILING_LINEAR` or `VK_IMAGE_TILING_OPTIMAL`, then `planeAspect` **must** be a single valid `multi-planar aspect mask` bit
- VUID-VkBindImagePlaneMemoryInfo-planeAspect-02284
If the image's `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `planeAspect` **must** be a single valid *memory plane* for the image (that is, `aspectMask` **must** specify a plane index that is less than the `VkDrmFormatModifierPropertiesEXT::drmFormatModifierPlaneCount` associated with the image's `format` and `VkImageDrmFormatModifierPropertiesEXT::drmFormatModifier`)

Valid Usage (Implicit)

- VUID-VkBindImagePlaneMemoryInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_BIND_IMAGE_PLANE_MEMORY_INFO`
- VUID-VkBindImagePlaneMemoryInfo-planeAspect-parameter
`planeAspect` **must** be a valid `VkImageAspectFlagBits` value

Buffer-Image Granularity

The implementation-dependent limit `bufferImageGranularity` specifies a page-like granularity at which linear and non-linear resources **must** be placed in adjacent memory locations to avoid aliasing. Two resources which do not satisfy this granularity requirement are said to *alias*. `bufferImageGranularity` is specified in bytes, and **must** be a power of two. Implementations which do not impose a granularity restriction **may** report a `bufferImageGranularity` value of one.

NOTE

Despite its name, `bufferImageGranularity` is really a granularity between “linear” and “non-linear” resources.

Given resourceA at the lower memory offset and resourceB at the higher memory offset in the same `VkDeviceMemory` object, where one resource is linear and the other is non-linear (as defined in the [Glossary](#)), and the following:

```
resourceA.end      = resourceA.memoryOffset + resourceA.size - 1
resourceA.endPage  = resourceA.end & ~(bufferImageGranularity-1)
resourceB.start    = resourceB.memoryOffset
resourceB.startPage = resourceB.start & ~(bufferImageGranularity-1)
```

The following property **must** hold:

```
resourceA.endPage < resourceB.startPage
```

That is, the end of the first resource (A) and the beginning of the second resource (B) **must** be on separate “pages” of size `bufferImageGranularity`. `bufferImageGranularity` **may** be different than the physical page size of the memory heap. This restriction is only needed when a linear resource and a non-linear resource are adjacent in memory and will be used simultaneously. The memory ranges of adjacent resources **can** be closer than `bufferImageGranularity`, provided they meet the *alignment* requirement for the objects in question.

Sparse block size in bytes and sparse image and buffer memory alignments **must** all be multiples of the `bufferImageGranularity`. Therefore, memory bound to sparse resources naturally satisfies the `bufferImageGranularity`.

12.8. Resource Sharing Mode

Buffer and image objects are created with a *sharing mode* controlling how they **can** be accessed

from queues. The supported sharing modes are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSharingMode {
    VK_SHARING_MODE_EXCLUSIVE = 0,
    VK_SHARING_MODE_CONCURRENT = 1,
} VkSharingMode;
```

- **VK_SHARING_MODE_EXCLUSIVE** specifies that access to any range or image subresource of the object will be exclusive to a single queue family at a time.
- **VK_SHARING_MODE_CONCURRENT** specifies that concurrent access to any range or image subresource of the object from multiple queue families is supported.

NOTE

VK_SHARING_MODE_CONCURRENT **may** result in lower performance access to the buffer or image than **VK_SHARING_MODE_EXCLUSIVE**.

Ranges of buffers and image subresources of image objects created using **VK_SHARING_MODE_EXCLUSIVE** **must** only be accessed by queues in the queue family that has *ownership* of the resource. Upon creation, such resources are not owned by any queue family; ownership is implicitly acquired upon first use within a queue. Once a resource using **VK_SHARING_MODE_EXCLUSIVE** is owned by some queue family, the application **must** perform a [queue family ownership transfer](#) if it wishes to make the memory contents of a range or image subresource accessible to a different queue family. **VK_SHARING_MODE_EXCLUSIVE** resources that are already owned by a queue family **may** be acquired by a different queue family without a queue family ownership transfer, but their contents become undefined.

NOTE

Before being used on the first queue, images still require a [layout transition](#) from these layouts:

- **VK_IMAGE_LAYOUT_UNDEFINED**
- **VK_IMAGE_LAYOUT_PREINITIALIZED**

A queue family **can** take ownership of an image subresource or buffer range of a resource created with **VK_SHARING_MODE_EXCLUSIVE**, without an ownership transfer, in the same way as for a resource that was just created; however, taking ownership in this way has the effect that the contents of the image subresource or buffer range are undefined.

Ranges of buffers and image subresources of image objects created using **VK_SHARING_MODE_CONCURRENT** **must** only be accessed by queues from the queue families specified through the **queueFamilyIndexCount** and **pQueueFamilyIndices** members of the corresponding create info structures.

12.8.1. External Resource Sharing

Resources **should** only be accessed in the Vulkan instance that has exclusive ownership of their underlying memory. Only one Vulkan instance has exclusive ownership of a resource's underlying memory at a given time, regardless of whether the resource was created using

`VK_SHARING_MODE_EXCLUSIVE` or `VK_SHARING_MODE_CONCURRENT`. Applications can transfer ownership of a resource's underlying memory only if the memory has been imported from or exported to another instance or external API using external memory handles. The semantics for transferring ownership outside of the instance are similar to those used for transferring ownership of `VK_SHARING_MODE_EXCLUSIVE` resources between queues, and is also accomplished using `VkBufferMemoryBarrier` or `VkImageMemoryBarrier` operations. To make the contents of the underlying memory accessible in the destination instance or API, applications **must**

1. Release exclusive ownership from the source instance or API.
2. Ensure the release operation has completed using semaphores or fences.
3. Acquire exclusive ownership in the destination instance or API

Unlike queue family ownership transfers, the destination instance or API is not specified explicitly when releasing ownership, nor is the source instance or API specified when acquiring ownership. Instead, the image or memory barrier's `dstQueueFamilyIndex` or `srcQueueFamilyIndex` parameters are set to the reserved queue family index `VK_QUEUE_FAMILY_EXTERNAL` or `VK_QUEUE_FAMILY_FOREIGN_EXT` to represent the external destination or source respectively.

Binding a resource to a memory object shared between multiple Vulkan instances or other APIs does not change the ownership of the underlying memory. The first entity to access the resource implicitly acquires ownership. An entity **can** also implicitly take ownership from another entity in the same way without an explicit ownership transfer. However, taking ownership in this way has the effect that the contents of the underlying memory are undefined.

Accessing a resource backed by memory that is owned by a particular instance or API has the same semantics as accessing a `VK_SHARING_MODE_EXCLUSIVE` resource, with one exception: Implementations **must** ensure layout transitions performed on one member of a set of identical subresources of identical images that alias the same range of an underlying memory object affect the layout of all the subresources in the set.

As a corollary, writes to any image subresources in such a set **must** not make the contents of memory used by other subresources in the set undefined. An application **can** define the content of a subresource of one image by performing device writes to an identical subresource of another image provided both images are bound to the same region of external memory. Applications **may** also add resources to such a set after the content of the existing set members has been defined without making the content undefined by creating a new image with the initial layout `VK_IMAGE_LAYOUT_UNDEFINED` and binding it to the same region of external memory as the existing images.

NOTE

Because layout transitions apply to all identical images aliasing the same region of external memory, the actual layout of the memory backing a new image as well as an existing image with defined content will not be undefined. Such an image is not usable until it acquires ownership of its memory from the existing owner. Therefore, the layout specified as part of this transition will be the true initial layout of the image. The undefined layout specified when creating it is a placeholder to simplify valid usage requirements.

12.9. Memory Aliasing

A range of a `VkDeviceMemory` allocation is *aliased* if it is bound to multiple resources simultaneously, as described below, via `vkBindImageMemory`, `vkBindBufferMemory`, or by binding the memory to resources in multiple Vulkan instances or external APIs using external memory handle export and import mechanisms.

Consider two resources, `resourceA` and `resourceB`, bound respectively to memory `rangeA` and `rangeB`. Let `paddedRangeA` and `paddedRangeB` be, respectively, `rangeA` and `rangeB` aligned to `bufferImageGranularity`. If the resources are both linear or both non-linear (as defined in the [Glossary](#)), then the resources *alias* the memory in the intersection of `rangeA` and `rangeB`. If one resource is linear and the other is non-linear, then the resources *alias* the memory in the intersection of `paddedRangeA` and `paddedRangeB`.

Applications **can** alias memory, but use of multiple aliases is subject to several constraints.

NOTE

Memory aliasing **can** be useful to reduce the total device memory footprint of an application, if some large resources are used for disjoint periods of time.

When a [non-linear](#), non-`VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` image is bound to an aliased range, all image subresources of the image *overlap* the range. When a linear image is bound to an aliased range, the image subresources that (according to the image's advertised layout) include bytes from the aliased range overlap the range. When a `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` image has sparse image blocks bound to an aliased range, only image subresources including those sparse image blocks overlap the range, and when the memory bound to the image's mip tail overlaps an aliased range all image subresources in the mip tail overlap the range.

Buffers, and linear image subresources are considered *host-accessible subresources* when they are in any of these layouts:

- `VK_IMAGE_LAYOUT_PREINITIALIZED`
- `VK_IMAGE_LAYOUT_GENERAL`

That is, the host has a well-defined addressing scheme to interpret the contents, and thus the layout of the data in memory **can** be consistently interpreted across aliases if each of those aliases is a host-accessible subresource. Non-linear images, and linear image subresources in other layouts, are not host-accessible.

If two aliases are both host-accessible, then they interpret the contents of the memory in consistent ways, and data written to one alias **can** be read by the other alias.

If two aliases are both images that were created with identical creation parameters, both were created with the `VK_IMAGE_CREATE_ALIAS_BIT` flag set, and both are bound identically to memory except for `VkBindImageMemoryDeviceGroupInfo::pDeviceIndices` and `VkBindImageMemoryDeviceGroupInfo::pSplitInstanceBindRegions`, then they interpret the contents of the memory in consistent ways, and data written to one alias **can** be read by the other alias.

Additionally, if an individual plane of a multi-planar image and a single-plane image alias the same memory, then they also interpret the contents of the memory in consistent ways under the same

conditions, but with the following modifications:

- Both **must** have been created with the `VK_IMAGE_CREATE_DISJOINT_BIT` flag.
- The single-plane image **must** have a `VkFormat` that is **equivalent** to that of the multi-planar image's individual plane.
- The single-plane image and the individual plane of the multi-planar image **must** be bound identically to memory except for `VkBindImageMemoryDeviceGroupInfo::pDeviceIndices` and `VkBindImageMemoryDeviceGroupInfo::pSplitInstanceBindRegions`.
- The **width** and **height** of the single-plane image are derived from the multi-planar image's dimensions in the manner listed for **plane compatibility** for the aliased plane.
- If either image's **tiling** is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then both images **must** be **linear**.
- All other creation parameters **must** be identical

Aliases created by binding the same memory to resources in multiple Vulkan instances or external APIs using external memory handle export and import mechanisms interpret the contents of the memory in consistent ways, and data written to one alias **can** be read by the other alias.

Otherwise, the aliases interpret the contents of the memory differently, and writes via one alias make the contents of memory partially or completely undefined to the other alias. If the first alias is a host-accessible subresource, then the bytes affected are those written by the memory operations according to its addressing scheme. If the first alias is not host-accessible, then the bytes affected are those overlapped by the image subresources that were written. If the second alias is a host-accessible subresource, the affected bytes become undefined. If the second alias is not host-accessible, all sparse image blocks (for sparse partially-resident images) or all image subresources (for non-sparse image and fully resident sparse images) that overlap the affected bytes become undefined.

If any image subresources are made undefined due to writes to an alias, then each of those image subresources **must** have its layout transitioned from `VK_IMAGE_LAYOUT_UNDEFINED` to a valid layout before it is used, or from `VK_IMAGE_LAYOUT_PREINITIALIZED` if the memory has been written by the host. If any sparse blocks of a sparse image have been made undefined, then only the image subresources containing them **must** be transitioned.

Use of an overlapping range by two aliases **must** be separated by a memory dependency using the appropriate **access types** if at least one of those uses performs writes, whether the aliases interpret memory consistently or not. If buffer or image memory barriers are used, the scope of the barrier **must** contain the entire range and/or set of image subresources that overlap.

If two aliasing image views are used in the same framebuffer, then the render pass **must** declare the attachments using the `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT`, and follow the other rules listed in that section.

NOTE

Memory recycled via an application suballocator (i.e. without freeing and reallocating the memory objects) is not substantially different from memory aliasing. However, a suballocator usually waits on a fence before recycling a region of memory, and signaling a fence involves sufficient implicit dependencies to satisfy

all the above requirements.

12.9.1. Resource Memory Overlap

Applications **can** safely access resources concurrently via separate device and host operations as long as the accessed memory locations are guaranteed to not overlap, as defined in [Memory Location](#), and the operation, resource, and access are otherwise independently valid.

Some operations have alignment requirements or access ambiguous [memory locations](#), so the semantics of a particular operation **should** be considered when determining the overlap. Such requirements will be described alongside the operation. Operations between host and device when using non-coherent memory are aligned to [nonCoherentAtomSize](#), as defined by [vkFlushMappedMemoryRanges](#) and [vkInvalidateMappedMemoryRanges](#).

NOTE

The intent is that buffers (or linear images) can be accessed concurrently, even when they share cache lines, but otherwise do not access the same memory range. The concept of a device cache line size is not exposed in the memory model.

Chapter 13. Images

Images are specialized resources that have multi-dimensional access rather than the typical linear access to memory. These resources allow implementations to optimize the memory layout for common access patterns, by mapping multi-dimensional coordinates to an implementation-dependent offset in the underlying memory.

Additionally, images are homogeneous, with every discrete coordinate associated with data that is in the same format as the data associated with any other coordinate. Each set of data associated with one of these coordinates is referred to in this specification as a *texel*. A texel can consist of up to 4 separate components, labeled as (R,G,B,A) in this chapter.

NOTE

This terminology is historical; texel is a combination of the words “**texture**” and “**element**”, and (R,G,B,A) are abbreviations for **Red**, **Green**, **Blue**, and **Alpha**. Early interactive computer graphics only supported operations that allowed image data to be used to add color “texture” to rendered objects, which is just a small subset of what the image operations on images described here enable. These terms are deeply embedded in the industry, and so are still used here by convention, despite images being used in far more varied ways than they were in the past.

Some image formats identify components as D for **depth**, S for **stencil**, or X for padding elements. Image format components identified as D are treated as R, and S components are treated as G for the purpose of image accesses. X components are ignored when reading, and **may** be modified in implementation-dependent ways when writing to that texel.

The coordinates used to identify a texel are six-dimensional, made up of the following integer indices:

- x - The first spatial index
- y - The second spatial index
- z - The third spatial index
- layer - The array index for arrayed images
- sample - The sample index
- level - The detail level

Each image is constructed with a number of texels in each dimension, with the integer size in each dimension for that image identified as:

- width - The number of x indices
- height - The number of y indices
- depth - The number of z indices
- layers - The number of layers
- samples - The number of samples
- levels - The number of detail levels

Of these sizes, the number of levels is somewhat unique - each further level reduces the number of indices in each of the x, y, and z dimensions by half, according to these formulae:

- $\text{width}_{\text{level}} = \max(1, \lceil \text{width} / 2^{\text{level}} \rceil)$
- $\text{height}_{\text{level}} = \max(1, \lceil \text{height} / 2^{\text{level}} \rceil)$
- $\text{depth}_{\text{level}} = \max(1, \lceil \text{depth} / 2^{\text{level}} \rceil)$

An image will have no more than one level where all of the above dimensions are set to 1. See [Image Mip Level Sizing](#) for more details.

The Vulkan specification allows the creation of resources with fewer dimensions than this (e.g. texel buffers); these can be considered equivalent to an image with all dimensions specified, with the missing dimensions having a size of 1, and an implicit coordinate value of 0.

13.1. Image Coordinate Validation

When accessing an image, a set of (x,y,z,layer,sample,level) coordinates are used to indicate which texel is accessed. These coordinates are first checked to see if they refer to texels within the image dimensions, according to the following equations:

- $x < \text{width}_{\text{level}}$
- $y < \text{height}_{\text{level}}$
- $z < \text{depth}_{\text{level}}$
- $\text{layer} < \text{layers}$
- $\text{sample} < \text{samples}$
- $\text{level} < \text{levels}$

If any of these equations evaluates to false, the coordinate is considered [out of bounds](#), otherwise they are *in bounds*.

13.2. Image Reads

Image reads use a set of (x,y,z,layer,sample,level) coordinates, validated as per [Image Coordinate Validation](#), and return a converted value for the texel at that coordinate. If the coordinates are out of bounds, behavior of the read is as described in [Shader Out-of-Bounds Memory Access](#). Otherwise, the read proceeds as follows.

13.2.1. Texel Decode

The formatted value of the texel at the (x,y,z,layer,sample,level) coordinate is read and decoded according to the procedures outlined in the [Khronos Data Format Specification](#).

For sRGB formats, the (R,G,B,A) components are first converted as if they are **UNORM** formats, and then sRGB to linear conversion is performed on the converted (R,G,B) components, as described in the “sRGB EOTF” section of the [Khronos Data Format Specification](#).

13.2.2. Component Substitution

If after conversion, less than four of the (R,G,B,A) components are present, missing components are substituted by the components of (0,0,0,1) for missing (R,G,B,A) components, respectively.

13.2.3. Numeric Encoding

The values are encoded according to the bit width and [numeric format](#) of each component:

- Components with a fixed-point numeric format or with a floating-point numeric format and a bit width less than or equal to 32 are encoded into the [IEEE-754](#) binary32 format.
- Components with a floating-point numeric format and a bit width greater than 32 are encoded into the [IEEE-754](#) binary64 format.
- Components with an integer numeric format and a bit width less than or equal to 32 are directly encoded as 32-bit integer values with the same signedness.
- Components with an integer numeric format and a bit width greater than 32 are directly encoded as 64-bit integer values with the same signedness.

These values are then returned as the result of the image read.

13.3. Image Writes

Image writes also use a set of (x,y,z,layer,sample,level) coordinates, validated as per [Image Coordinate Validation](#), and a value to be written. If the coordinates are out of bounds, behavior of the write is as described in [Shader Out-of-Bounds Memory Access](#). Otherwise, an image write is performed as follows:

13.3.1. Texel Encode

If the image format is sRGB, a linear to sRGB conversion is applied to the (R,G,B) components of value as described in the “sRGB EOTF” section of the [Khronos Data Format Specification](#).

The converted (R,G,B) and original A values are then encoded to the image format according to the procedures outlined in the [Khronos Data Format Specification](#). Components not present in the image format are discarded.

The final value is then written to the texel at (x,y,z,layer,sample,level).

Chapter 14. Samplers

VkSampler objects represent the state of an image sampler which is used by the implementation to read image data and apply filtering and other transformations for the shader.

Samplers are represented by **VkSampler** handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSampler)
```

To create a sampler object, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateSampler(
    VkDevice                                device,
    const VkSamplerCreateInfo*              pCreateInfo,
    const VkAllocationCallbacks*            pAllocator,
    VkSampler*                              pSampler);
```

- **device** is the logical device that creates the sampler.
- **pCreateInfo** is a pointer to a **VkSamplerCreateInfo** structure specifying the state of the sampler object.
- **pAllocator** controls host memory allocation as described in the **Memory Allocation** chapter.
- **pSampler** is a pointer to a **VkSampler** handle in which the resulting sampler object is returned.

If **VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations** is **VK_TRUE**, **vkCreateSampler** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage

- VUID-vkCreateSampler-device-09668
device **must** support at least one queue family with one of the **VK_QUEUE_COMPUTE_BIT** or **VK_QUEUE_GRAPHICS_BIT** capabilities
- VUID-vkCreateSampler-maxSamplerAllocationCount-04110
There **must** be less than **VkPhysicalDeviceLimits::maxSamplerAllocationCount** **VkSampler** objects currently created on the device
- VUID-vkCreateSampler-device-05068
The number of samplers currently allocated from **device** plus 1 **must** be less than or equal to the total number of samplers requested via **VkDeviceObjectReservationCreateInfo::samplerRequestCount** specified when **device** was created

Valid Usage (Implicit)

- VUID-vkCreateSampler-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkCreateSampler-pCreateInfo-parameter
pCreateInfo **must** be a valid pointer to a valid [VkSamplerCreateInfo](#) structure
- VUID-vkCreateSampler-pAllocator-null
pAllocator **must** be **NULL**
- VUID-vkCreateSampler-pSampler-parameter
pSampler **must** be a valid pointer to a [VkSampler](#) handle

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The [VkSamplerCreateInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkSamplerCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkSamplerCreateFlags  flags;
    VkFilter              magFilter;
    VkFilter              minFilter;
    VkSamplerMipmapMode   mipmapMode;
    VkSamplerAddressMode   addressModeU;
    VkSamplerAddressMode   addressModeV;
    VkSamplerAddressMode   addressModeW;
    float                 mipLodBias;
    VkBool32              anisotropyEnable;
    float                 maxAnisotropy;
    VkBool32              compareEnable;
    VkCompareOp           compareOp;
    float                 minLod;
    float                 maxLod;
    VkBorderColor         borderColor;
    VkBool32              unnormalizedCoordinates;
} VkSamplerCreateInfo;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is `NULL` or a pointer to a structure extending this structure.
- **flags** is a bitmask of [VkSamplerCreateFlagBits](#) describing additional parameters of the sampler.
- **magFilter** is a [VkFilter](#) value specifying the magnification filter to apply to lookups.
- **minFilter** is a [VkFilter](#) value specifying the minification filter to apply to lookups.
- **mipmapMode** is a [VkSamplerMipmapMode](#) value specifying the mipmap filter to apply to lookups.
- **addressModeU** is a [VkSamplerAddressMode](#) value specifying the addressing mode for U coordinates outside [0,1).
- **addressModeV** is a [VkSamplerAddressMode](#) value specifying the addressing mode for V coordinates outside [0,1).
- **addressModeW** is a [VkSamplerAddressMode](#) value specifying the addressing mode for W coordinates outside [0,1).
- **mipLodBias** is the bias to be added to mipmap LOD calculation and bias provided by image sampling functions in SPIR-V, as described in the [LOD Operation](#) section.
- **anisotropyEnable** is `VK_TRUE` to enable anisotropic filtering, as described in the [Texel Anisotropic Filtering](#) section, or `VK_FALSE` otherwise.
- **maxAnisotropy** is the anisotropy value clamp used by the sampler when **anisotropyEnable** is `VK_TRUE`. If **anisotropyEnable** is `VK_FALSE`, **maxAnisotropy** is ignored.
- **compareEnable** is `VK_TRUE` to enable comparison against a reference value during lookups, or `VK_FALSE` otherwise.
 - Note: Some implementations will default to shader state if this member does not match.

- `compareOp` is a `VkCompareOp` value specifying the comparison operator to apply to fetched data before filtering as described in the [Depth Compare Operation](#) section.
- `minLod` is used to clamp the [minimum of the computed LOD value](#).
- `maxLod` is used to clamp the [maximum of the computed LOD value](#). To avoid clamping the maximum value, set `maxLod` to the constant `VK_LOD_CLAMP_NONE`.
- `borderColor` is a `VkBorderColor` value specifying the predefined border color to use.
- `unnormalizedCoordinates` controls whether to use unnormalized or normalized texel coordinates to address texels of the image. When `unnormalizedCoordinates` is `VK_TRUE`, the range of the image coordinates used to lookup the texel is in the range of zero to the image size in each dimension. When `unnormalizedCoordinates` is `VK_FALSE`, the range of image coordinates is zero to one.

When `unnormalizedCoordinates` is `VK_TRUE`, images the sampler is used with in the shader have the following requirements:

- The `viewType` **must** be either `VK_IMAGE_VIEW_TYPE_1D` or `VK_IMAGE_VIEW_TYPE_2D`.
- The image view **must** have a single layer and a single mip level.

When `unnormalizedCoordinates` is `VK_TRUE`, image built-in functions in the shader that use the sampler have the following requirements:

- The functions **must** not use projection.
- The functions **must** not use offsets.

NOTE

Mapping of OpenGL to Vulkan Filter Modes

`magFilter` values of `VK_FILTER_NEAREST` and `VK_FILTER_LINEAR` directly correspond to `GL_NEAREST` and `GL_LINEAR` magnification filters. `minFilter` and `mipmapMode` combine to correspond to the similarly named OpenGL minification filter of `GL_minFilter_MIPMAP_mipmapMode` (e.g. `minFilter` of `VK_FILTER_LINEAR` and `mipmapMode` of `VK_SAMPLER_MIPMAP_MODE_NEAREST` correspond to `GL_LINEAR_MIPMAP_NEAREST`).

There are no Vulkan filter modes that directly correspond to OpenGL minification filters of `GL_LINEAR` or `GL_NEAREST`, but they **can** be emulated using `VK_SAMPLER_MIPMAP_MODE_NEAREST`, `minLod` = 0, and `maxLod` = 0.25, and using `minFilter` = `VK_FILTER_LINEAR` or `minFilter` = `VK_FILTER_NEAREST`, respectively.

Note that using a `maxLod` of zero would cause [magnification](#) to always be performed, and the `magFilter` to always be used. This is valid, just not an exact match for OpenGL behavior. Clamping the maximum LOD to 0.25 allows the λ value to be non-zero and minification to be performed, while still always rounding down to the base level. If the `minFilter` and `magFilter` are equal, then using a `maxLod` of zero also works.

The maximum number of sampler objects which **can** be simultaneously created on a device is implementation-dependent and specified by the `maxSamplerAllocationCount` member of the `VkPhysicalDeviceLimits` structure.

NOTE

For historical reasons, if `maxSamplerAllocationCount` is exceeded, some implementations may return `VK_ERROR_TOO_MANY_OBJECTS`. Exceeding this limit will result in undefined behavior, and an application should not rely on the use of the returned error code in order to identify when the limit is reached.

Since `VkSampler` is a non-dispatchable handle type, implementations **may** return the same handle for sampler state vectors that are identical. In such cases, all such objects would only count once against the `maxSamplerAllocationCount` limit.

Valid Usage

- VUID-VkSamplerCreateInfo-mipLodBias-01069
The absolute value of `mipLodBias` **must** be less than or equal to `VkPhysicalDeviceLimits::maxSamplerLodBias`
- VUID-VkSamplerCreateInfo-maxLod-01973
`maxLod` **must** be greater than or equal to `minLod`
- VUID-VkSamplerCreateInfo-anisotropyEnable-01070
If the `samplerAnisotropy` feature is not enabled, `anisotropyEnable` **must** be `VK_FALSE`
- VUID-VkSamplerCreateInfo-anisotropyEnable-01071
If `anisotropyEnable` is `VK_TRUE`, `maxAnisotropy` **must** be between `1.0` and `VkPhysicalDeviceLimits::maxSamplerAnisotropy`, inclusive
- VUID-VkSamplerCreateInfo-minFilter-01645
If `sampler YCBCR conversion` is enabled and the `potential format features` of the sampler `YCBCR conversion` do not support `VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT`, `minFilter` and `magFilter` **must** be equal to the sampler `YCBCR conversion's` `chromaFilter`
- VUID-VkSamplerCreateInfo-unnormalizedCoordinates-01072
If `unnormalizedCoordinates` is `VK_TRUE`, `minFilter` and `magFilter` **must** be equal
- VUID-VkSamplerCreateInfo-unnormalizedCoordinates-01073
If `unnormalizedCoordinates` is `VK_TRUE`, `mipmapMode` **must** be `VK_SAMPLER_MIPMAP_MODE_NEAREST`
- VUID-VkSamplerCreateInfo-unnormalizedCoordinates-01074
If `unnormalizedCoordinates` is `VK_TRUE`, `minLod` and `maxLod` **must** be zero
- VUID-VkSamplerCreateInfo-unnormalizedCoordinates-01075
If `unnormalizedCoordinates` is `VK_TRUE`, `addressModeU` and `addressModeV` **must** each be either `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE` or `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER`
- VUID-VkSamplerCreateInfo-unnormalizedCoordinates-01076
If `unnormalizedCoordinates` is `VK_TRUE`, `anisotropyEnable` **must** be `VK_FALSE`
- VUID-VkSamplerCreateInfo-unnormalizedCoordinates-01077
If `unnormalizedCoordinates` is `VK_TRUE`, `compareEnable` **must** be `VK_FALSE`
- VUID-VkSamplerCreateInfo-addressModeU-01078
If any of `addressModeU`, `addressModeV` or `addressModeW` are `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER`, `borderColor` **must** be a valid `VkBorderColor` value

- VUID-VkSamplerCreateInfo-addressModeU-01646
If **sampler Y'C_BC_R conversion** is enabled, **addressModeU**, **addressModeV**, and **addressModeW** **must** be **VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE**, **anisotropyEnable** **must** be **VK_FALSE**, and **unnormalizedCoordinates** **must** be **VK_FALSE**
- VUID-VkSamplerCreateInfo-None-01647
If **sampler Y'C_BC_R conversion** is enabled and the **pNext** chain includes a **VkSamplerReductionModeCreateInfo** structure, then the sampler reduction mode **must** be **VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE**
- VUID-VkSamplerCreateInfo-pNext-06726
If the **samplerFilterMinmax** feature is not enabled and the **pNext** chain includes a **VkSamplerReductionModeCreateInfo** structure, then the sampler reduction mode **must** be **VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE**
- VUID-VkSamplerCreateInfo-addressModeU-01079
If the **samplerMirrorClampToEdge** feature is not enabled, and if the **VK_KHR_sampler_mirror_clamp_to_edge** extension is not enabled, **addressModeU**, **addressModeV** and **addressModeW** **must** not be **VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE**
- VUID-VkSamplerCreateInfo-compareEnable-01080
If **compareEnable** is **VK_TRUE**, **compareOp** **must** be a valid **VkCompareOp** value
- VUID-VkSamplerCreateInfo-magFilter-01081
If either **magFilter** or **minFilter** is **VK_FILTER_CUBIC_EXT**, **anisotropyEnable** **must** be **VK_FALSE**
- VUID-VkSamplerCreateInfo-compareEnable-01423
If **compareEnable** is **VK_TRUE**, the **reductionMode** member **must** be **VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE**
- VUID-VkSamplerCreateInfo-borderColor-04011
If **borderColor** is one of **VK_BORDER_COLOR_FLOAT_CUSTOM_EXT** or **VK_BORDER_COLOR_INT_CUSTOM_EXT**, then a **VkSamplerCustomBorderColorCreateInfoEXT** **must** be included in the **pNext** chain
- VUID-VkSamplerCreateInfo-customBorderColors-04085
If the **customBorderColors** feature is not enabled, **borderColor** **must** not be **VK_BORDER_COLOR_FLOAT_CUSTOM_EXT** or **VK_BORDER_COLOR_INT_CUSTOM_EXT**
- VUID-VkSamplerCreateInfo-borderColor-04442
If **borderColor** is one of **VK_BORDER_COLOR_FLOAT_CUSTOM_EXT** or **VK_BORDER_COLOR_INT_CUSTOM_EXT**, and **VkSamplerCustomBorderColorCreateInfoEXT::format** is not **VK_FORMAT_UNDEFINED**, **VkSamplerCustomBorderColorCreateInfoEXT::customBorderColor** **must** be within the range of values representable in **format**
- VUID-VkSamplerCreateInfo-None-04012
The maximum number of samplers with custom border colors which **can** be simultaneously created on a device is implementation-dependent and specified by the **maxCustomBorderColorSamplers** member of the **VkPhysicalDeviceCustomBorderColorPropertiesEXT** structure

Valid Usage (Implicit)

- VUID-VkSamplerCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO`
- VUID-VkSamplerCreateInfo-pNext-pNext
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkSamplerCustomBorderColorCreateInfoEXT`, `VkSamplerReductionModeCreateInfo`, or `VkSamplerYcbcrConversionInfo`
- VUID-VkSamplerCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkSamplerCreateInfo-flags-zeroBitmask
`flags` **must** be `0`
- VUID-VkSamplerCreateInfo-magFilter-parameter
`magFilter` **must** be a valid `VkFilter` value
- VUID-VkSamplerCreateInfo-minFilter-parameter
`minFilter` **must** be a valid `VkFilter` value
- VUID-VkSamplerCreateInfo-mipmapMode-parameter
`mipmapMode` **must** be a valid `VkSamplerMipmapMode` value
- VUID-VkSamplerCreateInfo-addressModeU-parameter
`addressModeU` **must** be a valid `VkSamplerAddressMode` value
- VUID-VkSamplerCreateInfo-addressModeV-parameter
`addressModeV` **must** be a valid `VkSamplerAddressMode` value
- VUID-VkSamplerCreateInfo-addressModeW-parameter
`addressModeW` **must** be a valid `VkSamplerAddressMode` value

`VK_LOD_CLAMP_NONE` is a special constant value used for `VkSamplerCreateInfo::maxLod` to indicate that maximum LOD clamping should not be performed.

```
#define VK_LOD_CLAMP_NONE 1000.0F
```

Bits which **can** be set in `VkSamplerCreateInfo::flags`, specifying additional parameters of a sampler, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSamplerCreateFlagBits {
} VkSamplerCreateFlagBits;
```

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkSamplerCreateFlags;
```

`VkSamplerCreateFlags` is a bitmask type for setting a mask of zero or more `VkSamplerCreateFlagBits`.

The `VkSamplerReductionModeCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkSamplerReductionModeCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkSamplerReductionMode reductionMode;
} VkSamplerReductionModeCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `reductionMode` is a `VkSamplerReductionMode` value controlling how texture filtering combines texel values.

If the `pNext` chain of `VkSamplerCreateInfo` includes a `VkSamplerReductionModeCreateInfo` structure, then that structure includes a mode controlling how texture filtering combines texel values.

If this structure is not present, `reductionMode` is considered to be `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`.

Valid Usage (Implicit)

- VUID-VkSamplerReductionModeCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SAMPLER_REDUCTION_MODE_CREATE_INFO`
- VUID-VkSamplerReductionModeCreateInfo-reductionMode-parameter
`reductionMode` **must** be a valid `VkSamplerReductionMode` value

Reduction modes are specified by `VkSamplerReductionMode`, which takes values:

```
// Provided by VK_VERSION_1_2
typedef enum VkSamplerReductionMode {
    VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE = 0,
    VK_SAMPLER_REDUCTION_MODE_MIN = 1,
    VK_SAMPLER_REDUCTION_MODE_MAX = 2,
} VkSamplerReductionMode;
```

- `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE` specifies that texel values are combined by computing a weighted average of values in the footprint, using weights as specified in [the image operations chapter](#).
- `VK_SAMPLER_REDUCTION_MODE_MIN` specifies that texel values are combined by taking the component-wise minimum of values in the footprint with non-zero weights.
- `VK_SAMPLER_REDUCTION_MODE_MAX` specifies that texel values are combined by taking the

component-wise maximum of values in the footprint with non-zero weights.

Possible values of the `VkSamplerCreateInfo::magFilter` and `minFilter` parameters, specifying filters used for texture lookups, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkFilter {
    VK_FILTER_NEAREST = 0,
    VK_FILTER_LINEAR = 1,
    // Provided by VK_EXT_filter_cubic
    VK_FILTER_CUBIC_EXT = 1000015000,
} VkFilter;
```

- `VK_FILTER_NEAREST` specifies nearest filtering.
- `VK_FILTER_LINEAR` specifies linear filtering.
- `VK_FILTER_CUBIC_EXT` specifies cubic filtering.

These filters are described in detail in [Texel Filtering](#).

Possible values of the `VkSamplerCreateInfo::mipmapMode`, specifying the mipmap mode used for texture lookups, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSamplerMipmapMode {
    VK_SAMPLER_MIPMAP_MODE_NEAREST = 0,
    VK_SAMPLER_MIPMAP_MODE_LINEAR = 1,
} VkSamplerMipmapMode;
```

- `VK_SAMPLER_MIPMAP_MODE_NEAREST` specifies nearest filtering.
- `VK_SAMPLER_MIPMAP_MODE_LINEAR` specifies linear filtering.

These modes are described in detail in [Texel Filtering](#).

Possible values of the `VkSamplerCreateInfo::addressMode*` parameters, specifying the behavior of sampling with coordinates outside the range [0,1] for the respective u, v, or w coordinate as defined in the [Wrapping Operation](#) section, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkSamplerAddressMode {
    VK_SAMPLER_ADDRESS_MODE_REPEAT = 0,
    VK_SAMPLER_ADDRESS_MODE_MIRRORED_REPEAT = 1,
    VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE = 2,
    VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER = 3,
    // Provided by VK_VERSION_1_2
    VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE = 4,
} VkSamplerAddressMode;
```

- `VK_SAMPLER_ADDRESS_MODE_REPEAT` specifies that the repeat wrap mode will be used.
- `VK_SAMPLER_ADDRESS_MODE_MIRRORED_REPEAT` specifies that the mirrored repeat wrap mode will be used.
- `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE` specifies that the clamp to edge wrap mode will be used.
- `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER` specifies that the clamp to border wrap mode will be used.
- `VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE` specifies that the mirror clamp to edge wrap mode will be used. This is only valid if the `samplerMirrorClampToEdge` feature is enabled, or if the `VK_KHR_sampler_mirror_clamp_to_edge` extension is enabled.

Comparison operators compare a *reference* and a *test* value, and return a true (“passed”) or false (“failed”) value depending on the comparison operator chosen. The supported operators are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCompareOp {
    VK_COMPARE_OP_NEVER = 0,
    VK_COMPARE_OP_LESS = 1,
    VK_COMPARE_OP_EQUAL = 2,
    VK_COMPARE_OP_LESS_OR_EQUAL = 3,
    VK_COMPARE_OP_GREATER = 4,
    VK_COMPARE_OP_NOT_EQUAL = 5,
    VK_COMPARE_OP_GREATER_OR_EQUAL = 6,
    VK_COMPARE_OP_ALWAYS = 7,
} VkCompareOp;
```

- `VK_COMPARE_OP_NEVER` specifies that the comparison always evaluates false.
- `VK_COMPARE_OP_LESS` specifies that the comparison evaluates *reference* < *test*.
- `VK_COMPARE_OP_EQUAL` specifies that the comparison evaluates *reference* = *test*.
- `VK_COMPARE_OP_LESS_OR_EQUAL` specifies that the comparison evaluates *reference* ≤ *test*.
- `VK_COMPARE_OP_GREATER` specifies that the comparison evaluates *reference* > *test*.
- `VK_COMPARE_OP_NOT_EQUAL` specifies that the comparison evaluates *reference* ≠ *test*.
- `VK_COMPARE_OP_GREATER_OR_EQUAL` specifies that the comparison evaluates *reference* ≥ *test*.
- `VK_COMPARE_OP_ALWAYS` specifies that the comparison always evaluates true.

Comparison operators are used for:

- The **Depth Compare Operation** operator for a sampler, specified by `VkSamplerCreateInfo::compareOp`.
- The stencil comparison operator for the **stencil test**, specified by `vkCmdSetStencilOpEXT::compareOp` or `VkStencilOpState::compareOp`.
- The **Depth Comparison** operator for the **depth test**, specified by `vkCmdSetDepthCompareOpEXT::depthCompareOp` or `VkPipelineDepthStencilStateCreateInfo::depthCompareOp`.

Each such use describes how the *reference* and *test* values for that comparison are determined.

Possible values of `VkSamplerCreateInfo::borderColor`, specifying the border color used for texture lookups, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkBorderColor {
    VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK = 0,
    VK_BORDER_COLOR_INT_TRANSPARENT_BLACK = 1,
    VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK = 2,
    VK_BORDER_COLOR_INT_OPAQUE_BLACK = 3,
    VK_BORDER_COLOR_FLOAT_OPAQUE_WHITE = 4,
    VK_BORDER_COLOR_INT_OPAQUE_WHITE = 5,
    // Provided by VK_EXT_custom_border_color
    VK_BORDER_COLOR_FLOAT_CUSTOM_EXT = 1000287003,
    // Provided by VK_EXT_custom_border_color
    VK_BORDER_COLOR_INT_CUSTOM_EXT = 1000287004,
} VkBorderColor;
```

- `VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK` specifies a transparent, floating-point format, black color.
- `VK_BORDER_COLOR_INT_TRANSPARENT_BLACK` specifies a transparent, integer format, black color.
- `VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK` specifies an opaque, floating-point format, black color.
- `VK_BORDER_COLOR_INT_OPAQUE_BLACK` specifies an opaque, integer format, black color.
- `VK_BORDER_COLOR_FLOAT_OPAQUE_WHITE` specifies an opaque, floating-point format, white color.
- `VK_BORDER_COLOR_INT_OPAQUE_WHITE` specifies an opaque, integer format, white color.
- `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT` specifies that a `VkSamplerCustomBorderColorCreateInfoEXT` structure is included in the `VkSamplerCreateInfo::pNext` chain containing the color data in floating-point format.
- `VK_BORDER_COLOR_INT_CUSTOM_EXT` specifies that a `VkSamplerCustomBorderColorCreateInfoEXT` structure is included in the `VkSamplerCreateInfo::pNext` chain containing the color data in integer format.

These colors are described in detail in [Border Replacement](#).

To destroy a sampler, call:

```
// Provided by VK_VERSION_1_0
void vkDestroySampler(
    VkDevice          device,
    VkSampler          sampler,
    const VkAllocationCallbacks* pAllocator);
```

- `device` is the logical device that destroys the sampler.
- `sampler` is the sampler to destroy.

- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage

- VUID-vkDestroySampler-sampler-01082
All submitted commands that refer to **sampler** **must** have completed execution

Valid Usage (Implicit)

- VUID-vkDestroySampler-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkDestroySampler-sampler-parameter
If **sampler** is not [VK_NULL_HANDLE](#), **sampler** **must** be a valid [VkSampler](#) handle
- VUID-vkDestroySampler-pAllocator-null
pAllocator **must** be [NULL](#)
- VUID-vkDestroySampler-sampler-parent
If **sampler** is a valid handle, it **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to **sampler** **must** be externally synchronized

14.1. Sampler Y'C_BC_R Conversion

To create a sampler with Y'C_BC_R conversion enabled, add a [VkSamplerYcbcrConversionInfo](#) structure to the **pNext** chain of the [VkSamplerCreateInfo](#) structure. To create a sampler Y'C_BC_R conversion, the **samplerYcbcrConversion** feature **must** be enabled. Conversion **must** be fixed at pipeline creation time, through use of a combined image sampler with an immutable sampler in [VkDescriptorSetLayoutBinding](#).

A [VkSamplerYcbcrConversionInfo](#) **must** be provided for samplers to be used with image views that access [VK_IMAGE_ASPECT_COLOR_BIT](#) if the format is one of the [formats that require a sampler Y'C_BC_R conversion](#).

The [VkSamplerYcbcrConversionInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkSamplerYcbcrConversionInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkSamplerYcbcrConversion conversion;
} VkSamplerYcbcrConversionInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `conversion` is a `VkSamplerYcbcrConversion` handle created with `vkCreateSamplerYcbcrConversion`.

Valid Usage (Implicit)

- VUID-VkSamplerYcbcrConversionInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_INFO`
- VUID-VkSamplerYcbcrConversionInfo-conversion-parameter
`conversion` **must** be a valid `VkSamplerYcbcrConversion` handle

A sampler Y'C_BC_R conversion is an opaque representation of a device-specific sampler Y'C_BC_R conversion description, represented as a `VkSamplerYcbcrConversion` handle:

```
// Provided by VK_VERSION_1_1
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSamplerYcbcrConversion)
```

To create a `VkSamplerYcbcrConversion`, call:

```
// Provided by VK_VERSION_1_1
VkResult vkCreateSamplerYcbcrConversion(
    VkDevice device,
    const VkSamplerYcbcrConversionCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator,
    VkSamplerYcbcrConversion* pYcbcrConversion);
```

- `device` is the logical device that creates the sampler Y'C_BC_R conversion.
- `pCreateInfo` is a pointer to a `VkSamplerYcbcrConversionCreateInfo` structure specifying the requested sampler Y'C_BC_R conversion.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pYcbcrConversion` is a pointer to a `VkSamplerYcbcrConversion` handle in which the resulting sampler Y'C_BC_R conversion is returned.

The interpretation of the configured sampler Y'C_BC_R conversion is described in more detail in [the description of sampler Y'C_BC_R conversion](#) in the [Image Operations](#) chapter.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateSamplerYcbcrConversion` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateSamplerYcbcrConversion-None-01648

The `samplerYcbcrConversion` feature **must** be enabled

- VUID-vkCreateSamplerYcbcrConversion-device-05068

The number of sampler conversions currently allocated from `device` plus 1 **must** be less than or equal to the total number of sampler conversions requested via `VkDeviceObjectReservationCreateInfo::samplerYcbcrConversionRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateSamplerYcbcrConversion-device-parameter `device` **must** be a valid `VkDevice` handle
- VUID-vkCreateSamplerYcbcrConversion-pCreateInfo-parameter `pCreateInfo` **must** be a valid pointer to a valid `VkSamplerYcbcrConversionCreateInfo` structure
- VUID-vkCreateSamplerYcbcrConversion-pAllocator-null `pAllocator` **must** be `NULL`
- VUID-vkCreateSamplerYcbcrConversion-pYcbcrConversion-parameter `pYcbcrConversion` **must** be a valid pointer to a `VkSamplerYcbcrConversion` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkSamplerYcbcrConversionCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkSamplerYcbcrConversionCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkFormat              format;
    VkSamplerYcbcrModelConversion ycbcrModel;
    VkSamplerYcbcrRange   ycbcrRange;
    VkComponentMapping     components;
    VkChromaLocation       xChromaOffset;
    VkChromaLocation       yChromaOffset;
    VkFilter               chromaFilter;
    VkBool32               forceExplicitReconstruction;
} VkSamplerYcbcrConversionCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `format` is the format of the image from which color information will be retrieved.
- `ycbcrModel` describes the color matrix for conversion between color models.
- `ycbcrRange` describes whether the encoded values have headroom and foot room, or whether the encoding uses the full numerical range.
- `components` applies a *swizzle* based on `VkComponentSwizzle` enums prior to range expansion and color model conversion.
- `xChromaOffset` describes the *sample location* associated with downsampled chroma components in the x dimension. `xChromaOffset` has no effect for formats in which chroma components are not downsampled horizontally.
- `yChromaOffset` describes the *sample location* associated with downsampled chroma components in the y dimension. `yChromaOffset` has no effect for formats in which the chroma components are not downsampled vertically.
- `chromaFilter` is the filter for chroma reconstruction.
- `forceExplicitReconstruction` **can** be used to ensure that reconstruction is done explicitly, if supported.

NOTE

Setting `forceExplicitReconstruction` to `VK_TRUE` **may** have a performance penalty on implementations where explicit reconstruction is not the default mode of operation.

If `format` supports `VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT` the `forceExplicitReconstruction` value behaves as if it were `VK_TRUE`.

Sampler Y'_{C_B}R_{C_R} conversion objects do not support *external format conversion* without additional extensions defining *external formats*.

Valid Usage

- VUID-VkSamplerYcbcrConversionCreateInfo-format-04061
format **must** represent unsigned normalized values (i.e. the format **must** be a UNORM format)
- VUID-VkSamplerYcbcrConversionCreateInfo-format-01650
The potential format features of the sampler Y'C_BC_R conversion **must** support VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT or VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT
- VUID-VkSamplerYcbcrConversionCreateInfo-xChromaOffset-01651
If the potential format features of the sampler Y'C_BC_R conversion do not support VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT, xChromaOffset and yChromaOffset **must** not be VK_CHROMA_LOCATION_COSITED_EVEN if the corresponding components are downsampled
- VUID-VkSamplerYcbcrConversionCreateInfo-xChromaOffset-01652
If the potential format features of the sampler Y'C_BC_R conversion do not support VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT, xChromaOffset and yChromaOffset **must** not be VK_CHROMA_LOCATION_MIDPOINT if the corresponding components are downsampled
- VUID-VkSamplerYcbcrConversionCreateInfo-components-02581
If the format has a _422 or _420 suffix, then components.g **must** be the identity swizzle
- VUID-VkSamplerYcbcrConversionCreateInfo-components-02582
If the format has a _422 or _420 suffix, then components.a **must** be the identity swizzle, VK_COMPONENT_SWIZZLE_ONE, or VK_COMPONENT_SWIZZLE_ZERO
- VUID-VkSamplerYcbcrConversionCreateInfo-components-02583
If the format has a _422 or _420 suffix, then components.r **must** be the identity swizzle or VK_COMPONENT_SWIZZLE_B
- VUID-VkSamplerYcbcrConversionCreateInfo-components-02584
If the format has a _422 or _420 suffix, then components.b **must** be the identity swizzle or VK_COMPONENT_SWIZZLE_R
- VUID-VkSamplerYcbcrConversionCreateInfo-components-02585
If the format has a _422 or _420 suffix, and if either components.r or components.b is the identity swizzle, both values **must** be the identity swizzle
- VUID-VkSamplerYcbcrConversionCreateInfo-ycbcrModel-01655
If ycbcrModel is not VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY, then components.r, components.g, and components.b **must** correspond to components of the format; that is, components.r, components.g, and components.b **must** not be VK_COMPONENT_SWIZZLE_ZERO or VK_COMPONENT_SWIZZLE_ONE, and **must** not correspond to a component containing zero or one as a consequence of component substitution
- VUID-VkSamplerYcbcrConversionCreateInfo-ycbcrRange-02748
If ycbcrRange is VK_SAMPLER_YCBCR_RANGE_ITU_NARROW then the R, G and B components obtained by applying the component swizzle to format **must** each have a bit-depth greater than or equal to 8
- VUID-VkSamplerYcbcrConversionCreateInfo-forceExplicitReconstruction-01656
If the potential format features of the sampler Y'C_BC_R conversion do not support

`VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT` `forceExplicitReconstruction` **must** be `VK_FALSE`

- VUID-VkSamplerYcbcrConversionCreateInfo-chromaFilter-01657
If the [potential format features](#) of the sampler $Y'C_B C_R$ conversion do not support `VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT`, `chromaFilter` **must** not be `VK_FILTER_LINEAR`

Valid Usage (Implicit)

- VUID-VkSamplerYcbcrConversionCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_CREATE_INFO`
- VUID-VkSamplerYcbcrConversionCreateInfo-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of [VkExternalFormatQNX](#)
- VUID-VkSamplerYcbcrConversionCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkSamplerYcbcrConversionCreateInfo-format-parameter
`format` **must** be a valid [VkFormat](#) value
- VUID-VkSamplerYcbcrConversionCreateInfo-ycbcrModel-parameter
`ycbcrModel` **must** be a valid [VkSamplerYcbcrModelConversion](#) value
- VUID-VkSamplerYcbcrConversionCreateInfo-ycbcrRange-parameter
`ycbcrRange` **must** be a valid [VkSamplerYcbcrRange](#) value
- VUID-VkSamplerYcbcrConversionCreateInfo-components-parameter
`components` **must** be a valid [VkComponentMapping](#) structure
- VUID-VkSamplerYcbcrConversionCreateInfo-xChromaOffset-parameter
`xChromaOffset` **must** be a valid [VkChromaLocation](#) value
- VUID-VkSamplerYcbcrConversionCreateInfo-yChromaOffset-parameter
`yChromaOffset` **must** be a valid [VkChromaLocation](#) value
- VUID-VkSamplerYcbcrConversionCreateInfo-chromaFilter-parameter
`chromaFilter` **must** be a valid [VkFilter](#) value

If `chromaFilter` is `VK_FILTER_NEAREST`, chroma samples are reconstructed to luma component resolution using nearest-neighbour sampling. Otherwise, chroma samples are reconstructed using interpolation. More details can be found in [the description of sampler \$Y'C_B C_R\$ conversion](#) in the [Image Operations](#) chapter.

[VkSamplerYcbcrModelConversion](#) defines the conversion from the source color model to the shader color model. Possible values are:

```
// Provided by VK_VERSION_1_1
typedef enum VkSamplerYcbcrModelConversion {
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY = 0,
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_IDENTITY = 1,
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_709 = 2,
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_601 = 3,
    VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_2020 = 4,
} VkSamplerYcbcrModelConversion;
```

- `VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY` specifies that the input values to the conversion are unmodified.
- `VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_IDENTITY` specifies no model conversion but the inputs are range expanded as for $Y'C_B C_R$.
- `VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_709` specifies the color model conversion from $Y'C_B C_R$ to R'G'B' defined in BT.709 and described in the “BT.709 $Y'C_B C_R$ conversion” section of the [Khronos Data Format Specification](#).
- `VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_601` specifies the color model conversion from $Y'C_B C_R$ to R'G'B' defined in BT.601 and described in the “BT.601 $Y'C_B C_R$ conversion” section of the [Khronos Data Format Specification](#).
- `VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_2020` specifies the color model conversion from $Y'C_B C_R$ to R'G'B' defined in BT.2020 and described in the “BT.2020 $Y'C_B C_R$ conversion” section of the [Khronos Data Format Specification](#).

In the `VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_*` color models, for the input to the sampler $Y'C_B C_R$ range expansion and model conversion:

- the Y (Y' luma) component corresponds to the G component of an RGB image.
- the CB (C_B or “U” blue color difference) component corresponds to the B component of an RGB image.
- the CR (C_R or “V” red color difference) component corresponds to the R component of an RGB image.
- the alpha component, if present, is not modified by color model conversion.

These rules reflect the mapping of components after the component swizzle operation (controlled by `VkSamplerYcbcrConversionCreateInfo::components`).

NOTE

For example, an “YUVA” 32-bit format comprising four 8-bit components can be implemented as `VK_FORMAT_R8G8B8A8_UNORM` with a component mapping:

- `components.a = VK_COMPONENT_SWIZZLE_IDENTITY`
- `components.r = VK_COMPONENT_SWIZZLE_B`
- `components.g = VK_COMPONENT_SWIZZLE_R`
- `components.b = VK_COMPONENT_SWIZZLE_G`

The [VkSamplerYcbcrRange](#) enum describes whether color components are encoded using the full range of numerical values or whether values are reserved for headroom and foot room. [VkSamplerYcbcrRange](#) is defined as:

```
// Provided by VK_VERSION_1_1
typedef enum VkSamplerYcbcrRange {
    VK_SAMPLER_YCBCR_RANGE_ITU_FULL = 0,
    VK_SAMPLER_YCBCR_RANGE_ITU_NARROW = 1,
} VkSamplerYcbcrRange;
```

- [VK_SAMPLER_YCBCR_RANGE_ITU_FULL](#) specifies that the full range of the encoded values are valid and interpreted according to the ITU “full range” quantization rules.
- [VK_SAMPLER_YCBCR_RANGE_ITU_NARROW](#) specifies that headroom and foot room are reserved in the numerical range of encoded values, and the remaining values are expanded according to the ITU “narrow range” quantization rules.

The formulae for these conversions is described in the [Sampler Y'C_BC_R Range Expansion](#) section of the [Image Operations](#) chapter.

No range modification takes place if [ycbcrModel](#) is [VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY](#); the [ycbcrRange](#) field of [VkSamplerYcbcrConversionCreateInfo](#) is ignored in this case.

The [VkChromaLocation](#) enum defines the location of downsampled chroma component samples relative to the luma samples, and is defined as:

```
// Provided by VK_VERSION_1_1
typedef enum VkChromaLocation {
    VK_CHROMA_LOCATION_COSITED_EVEN = 0,
    VK_CHROMA_LOCATION_MIDPOINT = 1,
} VkChromaLocation;
```

- [VK_CHROMA_LOCATION_COSITED_EVEN](#) specifies that downsampled chroma samples are aligned with luma samples with even coordinates.
- [VK_CHROMA_LOCATION_MIDPOINT](#) specifies that downsampled chroma samples are located half way between each even luma sample and the nearest higher odd luma sample.

To destroy a sampler Y'C_BC_R conversion, call:

```
// Provided by VK_VERSION_1_1
void vkDestroySamplerYcbcrConversion(
    VkDevice device,
    VkSamplerYcbcrConversion ycbcrConversion,
    const VkAllocationCallbacks* pAllocator);
```

- [device](#) is the logical device that destroys the Y'C_BC_R conversion.
- [ycbcrConversion](#) is the conversion to destroy.

- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage (Implicit)

- VUID-vkDestroySamplerYcbcrConversion-device-parameter `device` **must** be a valid `VkDevice` handle
- VUID-vkDestroySamplerYcbcrConversion-ycbcrConversion-parameter
If `ycbcrConversion` is not `VK_NULL_HANDLE`, `ycbcrConversion` **must** be a valid `VkSamplerYcbcrConversion` handle
- VUID-vkDestroySamplerYcbcrConversion-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkDestroySamplerYcbcrConversion-ycbcrConversion-parent
If `ycbcrConversion` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `ycbcrConversion` **must** be externally synchronized

In addition to the predefined border color values, applications **can** provide a custom border color value by including the `VkSamplerCustomBorderColorCreateInfoEXT` structure in the `VkSamplerCreateInfo::pNext` chain.

The `VkSamplerCustomBorderColorCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_custom_border_color
typedef struct VkSamplerCustomBorderColorCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkClearColorValue  customBorderColor;
    VkFormat            format;
} VkSamplerCustomBorderColorCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `customBorderColor` is a `VkClearColorValue` representing the desired custom sampler border color.
- `format` is a `VkFormat` representing the format of the sampled image view(s). This field may be `VK_FORMAT_UNDEFINED` if the `customBorderColorWithoutFormat` feature is enabled.

NOTE

If `format` is a depth/stencil format, the aspect is determined by the value of `VkSamplerCreateInfo::borderColor`. If `VkSamplerCreateInfo::borderColor` is `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT`, the depth aspect is considered. If

`VkSamplerCreateInfo::borderColor` is `VK_BORDER_COLOR_INT_CUSTOM_EXT`, the stencil aspect is considered.

If `format` is `VK_FORMAT_UNDEFINED`, the `VkSamplerCreateInfo::borderColor` is `VK_BORDER_COLOR_INT_CUSTOM_EXT`, and the sampler is used with an image with a stencil format, then the implementation **must** source the custom border color from either the first or second components of `VkSamplerCreateInfo::borderColor` and **should** source it from the first component.

Valid Usage

- VUID-VkSamplerCustomBorderColorCreateInfoEXT-format-07605
If `format` is not `VK_FORMAT_UNDEFINED` and `format` is not a depth/stencil format then the `VkSamplerCreateInfo::borderColor` type **must** match the sampled type of the provided `format`, as shown in the *SPIR-V Type* column of the [Interpretation of Numeric Format](#) table
- VUID-VkSamplerCustomBorderColorCreateInfoEXT-format-04014
If the `customBorderColorWithoutFormat` feature is not enabled then `format` **must** not be `VK_FORMAT_UNDEFINED`
- VUID-VkSamplerCustomBorderColorCreateInfoEXT-format-04015
If the sampler is used to sample an image view of `VK_FORMAT_B4G4R4A4_UNORM_PACK16`, `VK_FORMAT_B5G6R5_UNORM_PACK16`, or `VK_FORMAT_B5G5R5A1_UNORM_PACK16` format then `format` **must** not be `VK_FORMAT_UNDEFINED`

Valid Usage (Implicit)

- VUID-VkSamplerCustomBorderColorCreateInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SAMPLER_CUSTOM_BORDER_COLOR_CREATE_INFO_EXT`
- VUID-VkSamplerCustomBorderColorCreateInfoEXT-format-parameter
`format` **must** be a valid `VkFormat` value

Chapter 15. Resource Descriptors

A *descriptor* is an opaque data structure representing a shader resource such as a buffer, buffer view, image view, sampler, or combined image sampler. Descriptors are organized into *descriptor sets*, which are bound during command recording for use in subsequent drawing commands. The arrangement of content in each descriptor set is determined by a *descriptor set layout*, which determines what descriptors can be stored within it. The sequence of descriptor set layouts that **can** be used by a pipeline is specified in a *pipeline layout*. Each pipeline object **can** use up to `maxBoundDescriptorSets` (see [Limits](#)) descriptor sets.

Shaders access resources via variables decorated with a descriptor set and binding number that link them to a descriptor in a descriptor set. The shader interface mapping to bound descriptor sets is described in the [Shader Resource Interface](#) section.

Shaders **can** also access buffers without going through descriptors by using [Physical Storage Buffer Access](#) to access them through 64-bit addresses.

15.1. Descriptor Types

There are a number of different types of descriptor supported by Vulkan, corresponding to different resources or usage. The following sections describe the API definitions of each descriptor type. The mapping of each type to SPIR-V is listed in the [Shader Resource and Descriptor Type Correspondence](#) and [Shader Resource and Storage Class Correspondence](#) tables in the [Shader Interfaces](#) chapter.

15.1.1. Storage Image

A *storage image* (`VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`) is a descriptor type associated with an [image resource](#) via an [image view](#) that load, store, and atomic operations **can** be performed on.

Storage image loads are supported in all shader stages for image views whose [format features](#) contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT`.

Stores to storage images are supported in compute shaders for image views whose [format features](#) contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT`.

Atomic operations on storage images are supported in compute shaders for image views whose [format features](#) contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT`.

When the `fragmentStoresAndAtomics` feature is enabled, stores and atomic operations are also supported for storage images in fragment shaders with the same set of image formats as supported in compute shaders. When the `vertexPipelineStoresAndAtomics` feature is enabled, stores and atomic operations are also supported in vertex, tessellation, and geometry shaders with the same set of image formats as supported in compute shaders.

The image subresources for a storage image **must** be in the `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR` or `VK_IMAGE_LAYOUT_GENERAL` layout in order to access its data in a shader.

15.1.2. Sampler

A *sampler descriptor* (`VK_DESCRIPTOR_TYPE_SAMPLER`) is a descriptor type associated with a [sampler](#) object, used to control the behavior of [sampling operations](#) performed on a [sampled image](#).

15.1.3. Sampled Image

A *sampled image* (`VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`) is a descriptor type associated with an [image resource](#) via an [image view](#) that [sampling operations](#) **can** be performed on.

Shaders combine a sampled image variable and a sampler variable to perform sampling operations.

Sampled images are supported in all shader stages for image views whose [format features](#) contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`.

An image subresources for a sampled image **must** be in one of the following layouts:

- `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_GENERAL`
- `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`
- `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`
- `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`

15.1.4. Combined Image Sampler

A *combined image sampler* (`VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`) is a single descriptor type associated with both a [sampler](#) and an [image resource](#), combining both a [sampler](#) and [sampled image](#) descriptor into a single descriptor.

If the descriptor refers to a sampler that performs [Y'C_BC_R conversion](#), the sampler **must** only be used to sample the image in the same descriptor. Otherwise, the sampler and image in this type of descriptor **can** be used freely with any other samplers and images.

An image subresources for a combined image sampler **must** be in one of the following layouts:

- `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_GENERAL`
- `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`
- `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`

- `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`

NOTE

On some implementations, it **may** be more efficient to sample from an image using a combination of sampler and sampled image that are stored together in the descriptor set in a combined descriptor.

15.1.5. Uniform Texel Buffer

A *uniform texel buffer* (`VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER`) is a descriptor type associated with a [buffer resource](#) via a [buffer view](#) that [image sampling operations](#) **can** be performed on.

Uniform texel buffers define a tightly-packed 1-dimensional linear array of texels, with texels going through format conversion when read in a shader in the same way as they are for an image.

Load operations from uniform texel buffers are supported in all shader stages for buffer view formats which report [format features](#) support for `VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT`

15.1.6. Storage Texel Buffer

A *storage texel buffer* (`VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`) is a descriptor type associated with a [buffer resource](#) via a [buffer view](#) that [image load, store, and atomic operations](#) **can** be performed on.

Storage texel buffers define a tightly-packed 1-dimensional linear array of texels, with texels going through format conversion when read in a shader in the same way as they are for an image. Unlike [uniform texel buffers](#), these buffers can also be written to in the same way as for [storage images](#).

Storage texel buffer loads are supported in all shader stages for texel buffer view formats which report [format features](#) support for `VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT`

Stores to storage texel buffers are supported in compute shaders for texel buffer formats which report [format features](#) support for `VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT`

Atomic operations on storage texel buffers are supported in compute shaders for texel buffer formats which report [format features](#) support for `VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT`

When the `fragmentStoresAndAtomics` feature is enabled, stores and atomic operations are also supported for storage texel buffers in fragment shaders with the same set of texel buffer formats as supported in compute shaders. When the `vertexPipelineStoresAndAtomics` feature is enabled, stores and atomic operations are also supported in vertex, tessellation, and geometry shaders with the same set of texel buffer formats as supported in compute shaders.

15.1.7. Storage Buffer

A *storage buffer* (`VK_DESCRIPTOR_TYPE_STORAGE_BUFFER`) is a descriptor type associated with a [buffer resource](#) directly, described in a shader as a structure with various members that load, store, and atomic operations **can** be performed on.

NOTE

Atomic operations **can** only be performed on members of certain types as defined in the [SPIR-V environment appendix](#).

15.1.8. Uniform Buffer

A *uniform buffer* (`VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER`) is a descriptor type associated with a [buffer resource](#) directly, described in a shader as a structure with various members that load operations **can** be performed on.

15.1.9. Dynamic Uniform Buffer

A *dynamic uniform buffer* (`VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`) is almost identical to a [uniform buffer](#), and differs only in how the offset into the buffer is specified. The base offset calculated by the `VkDescriptorBufferInfo` when initially [updating the descriptor set](#) is added to a [dynamic offset](#) when binding the descriptor set.

15.1.10. Dynamic Storage Buffer

A *dynamic storage buffer* (`VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`) is almost identical to a [storage buffer](#), and differs only in how the offset into the buffer is specified. The base offset calculated by the `VkDescriptorBufferInfo` when initially [updating the descriptor set](#) is added to a [dynamic offset](#) when binding the descriptor set.

15.1.11. Input Attachment

An *input attachment* (`VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`) is a descriptor type associated with an [image resource](#) via an [image view](#) that **can** be used for [framebuffer local](#) load operations in fragment shaders.

All image formats that are supported for color attachments (`VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`) or depth/stencil attachments (`VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT`) for a given image tiling mode are also supported for input attachments.

An image view used as an input attachment **must** be in one of the following layouts:

- `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_GENERAL`
- `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`
- `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`

- `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`
- `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`

15.2. Descriptor Sets

Descriptors are grouped together into descriptor set objects. A descriptor set object is an opaque object containing storage for a set of descriptors, where the types and number of descriptors is defined by a descriptor set layout. The layout object **may** be used to define the association of each descriptor binding with memory or other implementation resources. The layout is used both for determining the resources that need to be associated with the descriptor set, and determining the interface between shader stages and shader resources.

15.2.1. Descriptor Set Layout

A descriptor set layout object is defined by an array of zero or more descriptor bindings. Each individual descriptor binding is specified by a descriptor type, a count (array size) of the number of descriptors in the binding, a set of shader stages that **can** access the binding, and (if using immutable samplers) an array of sampler descriptors.

Descriptor set layout objects are represented by `VkDescriptorSetLayout` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDescriptorSetLayout)
```

To create descriptor set layout objects, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateDescriptorSetLayout(
    VkDevice                                device,
    const VkDescriptorSetLayoutCreateInfo*  pCreateInfo,
    const VkAllocationCallbacks*           pAllocator,
    VkDescriptorSetLayout*                  pSetLayout);
```

- `device` is the logical device that creates the descriptor set layout.
- `pCreateInfo` is a pointer to a `VkDescriptorSetLayoutCreateInfo` structure specifying the state of the descriptor set layout object.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pSetLayout` is a pointer to a `VkDescriptorSetLayout` handle in which the resulting descriptor set layout object is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateDescriptorSetLayout` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateDescriptorSetLayout-support-09582

If the descriptor layout exceeds the limits reported through the [physical device limits](#), then [vkGetDescriptorSetLayoutSupport](#) **must** have returned [VkDescriptorSetLayoutSupport](#) with [support](#) equal to [VK_TRUE](#) for [pCreateInfo](#)

- VUID-vkCreateDescriptorSetLayout-device-05068

The number of descriptor set layouts currently allocated from [device](#) plus 1 **must** be less than or equal to the total number of descriptor set layouts requested via [VkDeviceObjectReservationCreateInfo::descriptorSetLayoutRequestCount](#) specified when [device](#) was created

- VUID-vkCreateDescriptorSetLayout-layoutbindings-device-05089

The number of descriptor set layout bindings currently allocated from [device](#) across all [VkDescriptorSetLayout](#) objects plus [pCreateInfo->bindingCount](#) **must** be less than or equal to the total number of descriptor set layout bindings requested via [VkDeviceObjectReservationCreateInfo::descriptorSetLayoutBindingRequestCount](#) specified when [device](#) was created

Valid Usage (Implicit)

- VUID-vkCreateDescriptorSetLayout-device-parameter

[device](#) **must** be a valid [VkDevice](#) handle

- VUID-vkCreateDescriptorSetLayout-pCreateInfo-parameter

[pCreateInfo](#) **must** be a valid pointer to a valid [VkDescriptorSetLayoutCreateInfo](#) structure

- VUID-vkCreateDescriptorSetLayout-pAllocator-null

[pAllocator](#) **must** be [NULL](#)

- VUID-vkCreateDescriptorSetLayout-pSetLayout-parameter

[pSetLayout](#) **must** be a valid pointer to a [VkDescriptorSetLayout](#) handle

Return Codes

Success

- [VK_SUCCESS](#)

Failure

- [VK_ERROR_OUT_OF_DEVICE_MEMORY](#)
- [VK_ERROR_OUT_OF_HOST_MEMORY](#)
- [VK_ERROR_UNKNOWN](#)
- [VK_ERROR_VALIDATION_FAILED](#)

Information about the descriptor set layout is passed in a [VkDescriptorSetLayoutCreateInfo](#)

structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorSetLayoutCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkDescriptorSetLayoutCreateFlags flags;
    uint32_t                 bindingCount;
    const VkDescriptorSetLayoutBinding* pBindings;
} VkDescriptorSetLayoutCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask specifying options for descriptor set layout creation.
- `bindingCount` is the number of elements in `pBindings`.
- `pBindings` is a pointer to an array of `VkDescriptorSetLayoutBinding` structures.

Valid Usage

- VUID-VkDescriptorSetLayoutCreateInfo-binding-00279
The `VkDescriptorSetLayoutBinding::binding` members of the elements of the `pBindings` array **must** each have different values
- VUID-VkDescriptorSetLayoutCreateInfo-flags-03000
If any binding has the `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` bit set, `flags` **must** include `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT`
- VUID-VkDescriptorSetLayoutCreateInfo-descriptorType-03001
If any binding has the `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` bit set, then all bindings **must** not have `descriptorType` of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`
- VUID-VkDescriptorSetLayoutCreateInfo-bindingCount-05011
`bindingCount` **must** be less than or equal to `maxDescriptorSetLayoutBindings`
- VUID-VkDescriptorSetLayoutCreateInfo-descriptorCount-05071
The sum of `descriptorCount` over all bindings in `pBindings` that have `descriptorType` of `VK_DESCRIPTOR_TYPE_SAMPLER` or `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` and `pImmutableSamplers` not equal to `NULL` **must** be less than or equal to `VkDeviceObjectReservationCreateInfo::maxImmutableSamplersPerDescriptorSetLayout`

Valid Usage (Implicit)

- VUID-VkDescriptorSetLayoutCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO`
- VUID-VkDescriptorSetLayoutCreateInfo-pNext-pNext

`pNext` **must** be `NULL` or a pointer to a valid instance of `VkDescriptorSetLayoutBindingFlagsCreateInfo`

- VUID-VkDescriptorSetLayoutCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkDescriptorSetLayoutCreateInfo-flags-parameter
`flags` **must** be a valid combination of `VkDescriptorSetLayoutCreateFlagBits` values
- VUID-VkDescriptorSetLayoutCreateInfo-pBindings-parameter
If `bindingCount` is not 0, `pBindings` **must** be a valid pointer to an array of `bindingCount` valid `VkDescriptorSetLayoutBinding` structures

Bits which **can** be set in `VkDescriptorSetLayoutCreateInfo::flags`, specifying options for descriptor set layout, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkDescriptorSetLayoutCreateFlagBits {
    // Provided by VK_VERSION_1_2
    VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT = 0x00000002,
} VkDescriptorSetLayoutCreateFlagBits;
```

- `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` specifies that descriptor sets using this layout **must** be allocated from a descriptor pool created with the `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT` bit set. Descriptor set layouts created with this bit set have alternate limits for the maximum number of descriptors per-stage and per-pipeline layout. The non-UpdateAfterBind limits only count descriptors in sets created without this flag. The UpdateAfterBind limits count all descriptors, but the limits **may** be higher than the non-UpdateAfterBind limits.

NOTE

All bits for this type are defined by extensions, and none of those extensions are enabled in this build of the specification.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkDescriptorSetLayoutCreateFlags;
```

`VkDescriptorSetLayoutCreateFlags` is a bitmask type for setting a mask of zero or more `VkDescriptorSetLayoutCreateFlagBits`.

The `VkDescriptorSetLayoutBinding` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorSetLayoutBinding {
    uint32_t          binding;
    VkDescriptorType   descriptorType;
    uint32_t          descriptorCount;
    VkShaderStageFlags stageFlags;
    const VkSampler*   pImmutableSamplers;
} VkDescriptorSetLayoutBinding;
```

- **binding** is the binding number of this entry and corresponds to a resource of the same binding number in the shader stages.
- **descriptorType** is a **VkDescriptorType** specifying which type of resource descriptors are used for this binding.
- **descriptorCount** is the number of descriptors contained in the binding, accessed in a shader as an array. If **descriptorCount** is zero this binding entry is reserved and the resource **must** not be accessed from any stage via this binding within any pipeline using the set layout.
- **stageFlags** member is a bitmask of **VkShaderStageFlagBits** specifying which pipeline shader stages **can** access a resource for this binding. **VK_SHADER_STAGE_ALL** is a shorthand specifying that all defined shader stages, including any additional stages defined by extensions, **can** access the resource.

If a shader stage is not included in **stageFlags**, then a resource **must** not be accessed from that stage via this binding within any pipeline using the set layout. Other than input attachments which are limited to the fragment shader, there are no limitations on what combinations of stages **can** use a descriptor binding, and in particular a binding **can** be used by both graphics stages and the compute stage.

- **pImmutableSamplers** affects initialization of samplers. If **descriptorType** specifies a **VK_DESCRIPTOR_TYPE_SAMPLER** or **VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER** type descriptor, then **pImmutableSamplers** **can** be used to initialize a set of *immutable samplers*. Immutable samplers are permanently bound into the set layout and **must** not be changed; updating a **VK_DESCRIPTOR_TYPE_SAMPLER** descriptor with immutable samplers is not allowed and updates to a **VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER** descriptor with immutable samplers does not modify the samplers (the image views are updated, but the sampler updates are ignored). If **pImmutableSamplers** is not **NULL**, then it is a pointer to an array of sampler handles that will be copied into the set layout and used for the corresponding binding. Only the sampler handles are copied; the sampler objects **must** not be destroyed before the final use of the set layout and any descriptor pools and sets created using it. If **pImmutableSamplers** is **NULL**, then the sampler slots are dynamic and sampler handles **must** be bound into descriptor sets using this layout. If **descriptorType** is not one of these descriptor types, then **pImmutableSamplers** is ignored.

The above layout definition allows the descriptor bindings to be specified sparsely such that not all binding numbers between 0 and the maximum binding number need to be specified in the **pBindings** array. Bindings that are not specified have a **descriptorCount** and **stageFlags** of zero, and the value of **descriptorType** is undefined. However, all binding numbers between 0 and the maximum binding number in the **VkDescriptorSetLayoutCreateInfo::pBindings** array **may** consume memory in the descriptor set layout even if not all descriptor bindings are used, though it **should**

not consume additional memory from the descriptor pool.

NOTE

The maximum binding number specified **should** be as compact as possible to avoid wasted memory.

Valid Usage

- VUID-VkDescriptorSetLayoutBinding-descriptorType-00282
If `descriptorType` is `VK_DESCRIPTOR_TYPE_SAMPLER` or `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, and `descriptorCount` is not 0 and `pImmutableSamplers` is not NULL, `pImmutableSamplers` **must** be a valid pointer to an array of `descriptorCount` valid `VkSampler` handles
- VUID-VkDescriptorSetLayoutBinding-descriptorCount-09465
If `descriptorCount` is not 0, `stageFlags` **must** be `VK_SHADER_STAGE_ALL` or a valid combination of other `VkShaderStageFlagBits` values
- VUID-VkDescriptorSetLayoutBinding-descriptorType-01510
If `descriptorType` is `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` and `descriptorCount` is not 0, then `stageFlags` **must** be 0 or `VK_SHADER_STAGE_FRAGMENT_BIT`
- VUID-VkDescriptorSetLayoutBinding-pImmutableSamplers-04009
The sampler objects indicated by `pImmutableSamplers` **must** not have a `borderColor` with one of the values `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT` or `VK_BORDER_COLOR_INT_CUSTOM_EXT`
- VUID-VkDescriptorSetLayoutBinding-binding-05012
`binding` **must** be less than the value of `VkDeviceObjectReservationCreateInfo::descriptorSetLayoutBindingLimit` provided when the device was created

Valid Usage (Implicit)

- VUID-VkDescriptorSetLayoutBinding-descriptorType-parameter
`descriptorType` **must** be a valid `VkDescriptorType` value

If the `pNext` chain of a `VkDescriptorSetLayoutCreateInfo` structure includes a `VkDescriptorSetLayoutBindingFlagsCreateInfo` structure, then that structure includes an array of flags, one for each descriptor set layout binding.

The `VkDescriptorSetLayoutBindingFlagsCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkDescriptorSetLayoutBindingFlagsCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           bindingCount;
    const VkDescriptorBindingFlags* pBindingFlags;
} VkDescriptorSetLayoutBindingFlagsCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `bindingCount` is zero or the number of elements in `pBindingFlags`.
- `pBindingFlags` is a pointer to an array of `VkDescriptorBindingFlags` bitfields, one for each descriptor set layout binding.

If `bindingCount` is zero or if this structure is not included in the `pNext` chain, the `VkDescriptorBindingFlags` for each descriptor set layout binding is considered to be zero. Otherwise, the descriptor set layout binding at `VkDescriptorSetLayoutCreateInfo::pBindings[i]` uses the flags in `pBindingFlags[i]`.

Valid Usage

- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-bindingCount-03002
If `bindingCount` is not zero, `bindingCount` **must** equal `VkDescriptorSetLayoutCreateInfo::bindingCount`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-pBindingFlags-03004
If an element of `pBindingFlags` includes `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT`, then it **must** be the element with the highest `binding` number
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingUniformBufferUpdateAfterBind-03005
If `VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingUniformBufferUpdateAfterBind` is not enabled, all bindings with descriptor type `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` **must** not use `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingSampledImageUpdateAfterBind-03006
If `VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingSampledImageUpdateAfterBind` is not enabled, all bindings with descriptor type `VK_DESCRIPTOR_TYPE_SAMPLER`, `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, or `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE` **must** not use `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingStorageImageUpdateAfterBind-03007
If `VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingStorageImageUpdateAfterBind` is not enabled, all bindings with descriptor type `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE` **must** not use `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingStorageBufferUpdateAfterBind-03008
If `VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingStorageBufferUpdateAfterBind` is not enabled, all bindings with descriptor type `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` **must** not use

VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT

- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingUniformTexelBufferUpdateAfterBind-03009
If [VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingUniformTexelBufferUpdateAfterBind](#) is not enabled, all bindings with descriptor type `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` **must** not use `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingStorageTexelBufferUpdateAfterBind-03010
If [VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingStorageTexelBufferUpdateAfterBind](#) is not enabled, all bindings with descriptor type `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` **must** not use `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-None-03011
All bindings with descriptor type `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`, `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`, or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` **must** not use `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingUpdateUnusedWhilePending-03012
If [VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingUpdateUnusedWhilePending](#) is not enabled, all elements of `pBindingFlags` **must** not include `VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingPartiallyBound-03013
If [VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingPartiallyBound](#) is not enabled, all elements of `pBindingFlags` **must** not include `VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-descriptorBindingVariableDescriptorCount-03014
If [VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingVariableDescriptorCount](#) is not enabled, all elements of `pBindingFlags` **must** not include `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-pBindingFlags-03015
If an element of `pBindingFlags` includes `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT`, that element's `descriptorType` **must** not be `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`

Valid Usage (Implicit)

- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_BINDING_FLAGS_CREATE_INFO`
- VUID-VkDescriptorSetLayoutBindingFlagsCreateInfo-pBindingFlags-parameter

If `bindingCount` is not 0, `pBindingFlags` **must** be a valid pointer to an array of `bindingCount` valid combinations of `VkDescriptorBindingFlagBits` values

Bits which **can** be set in each element of `VkDescriptorSetLayoutBindingFlagsCreateInfo::pBindingFlags`, specifying options for the corresponding descriptor set layout binding, are:

```
// Provided by VK_VERSION_1_2
typedef enum VkDescriptorBindingFlagBits {
    VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT = 0x00000001,
    VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT = 0x00000002,
    VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT = 0x00000004,
    VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT = 0x00000008,
} VkDescriptorBindingFlagBits;
```

- `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` specifies that if descriptors in this binding are updated between when the descriptor set is bound in a command buffer and when that command buffer is submitted to a queue, then the submission will use the most recently set descriptors for this binding and the updates do not invalidate the command buffer. Descriptor bindings created with this flag are also partially exempt from the external synchronization requirement in `vkUpdateDescriptorSets`. Multiple descriptors with this flag set **can** be updated concurrently in different threads, though the same descriptor **must** not be updated concurrently by two threads. Descriptors with this flag set **can** be updated concurrently with the set being bound to a command buffer in another thread, but not concurrently with the set being reset or freed.
- `VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT` specifies that descriptors in this binding that are not *dynamically used* need not contain valid descriptors at the time the descriptors are consumed. A descriptor is dynamically used if any shader invocation executes an instruction that performs any memory access using the descriptor. If a descriptor is not dynamically used, any resource referenced by the descriptor is not considered to be referenced during command execution.
- `VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT` specifies that descriptors in this binding **can** be updated after a command buffer has bound this descriptor set, or while a command buffer that uses this descriptor set is pending execution, as long as the descriptors that are updated are not used by those command buffers. Descriptor bindings created with this flag are also partially exempt from the external synchronization requirement in `vkUpdateDescriptorSetWithTemplateKHR` and `vkUpdateDescriptorSets` in the same way as for `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`. If `VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT` is also set, then descriptors **can** be updated as long as they are not dynamically used by any shader invocations. If `VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT` is not set, then descriptors **can** be updated as long as they are not statically used by any shader invocations.
- `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT` specifies that this is a *variable-sized descriptor binding* whose size will be specified when a descriptor set is allocated using this layout. The value of `descriptorCount` is treated as an upper bound on the size of the binding. This **must** only be used for the last binding in the descriptor set layout (i.e. the binding with the largest value of `binding`). For the purposes of counting against limits such as `maxDescriptorSet*` and `maxPerStageDescriptor*`, the full value of `descriptorCount` is counted.

NOTE

Note that while `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` and `VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT` both involve updates to descriptor sets after they are bound, `VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT` is a weaker requirement since it is only about descriptors that are not used, whereas `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` requires the implementation to observe updates to descriptors that are used.

```
// Provided by VK_VERSION_1_2
typedef VkFlags VkDescriptorBindingFlags;
```

`VkDescriptorBindingFlags` is a bitmask type for setting a mask of zero or more `VkDescriptorBindingFlagBits`.

To query information about whether a descriptor set layout **can** be created, call:

```
// Provided by VK_VERSION_1_1
void vkGetDescriptorSetLayoutSupport(
    VkDevice device,
    const VkDescriptorSetLayoutCreateInfo* pCreateInfo,
    VkDescriptorSetLayoutSupport* pSupport);
```

- `device` is the logical device that would create the descriptor set layout.
- `pCreateInfo` is a pointer to a `VkDescriptorSetLayoutCreateInfo` structure specifying the state of the descriptor set layout object.
- `pSupport` is a pointer to a `VkDescriptorSetLayoutSupport` structure, in which information about support for the descriptor set layout object is returned.

Some implementations have limitations on what fits in a descriptor set which are not easily expressible in terms of existing limits like `maxDescriptorSet*`, for example if all descriptor types share a limited space in memory but each descriptor is a different size or alignment. This command returns information about whether a descriptor set satisfies this limit. If the descriptor set layout satisfies the `VkPhysicalDeviceMaintenance3Properties::maxPerSetDescriptors` limit, this command is guaranteed to return `VK_TRUE` in `VkDescriptorSetLayoutSupport::supported`. If the descriptor set layout exceeds the `VkPhysicalDeviceMaintenance3Properties::maxPerSetDescriptors` limit, whether the descriptor set layout is supported is implementation-dependent and **may** depend on whether the descriptor sizes and alignments cause the layout to exceed an internal limit.

This command does not consider other limits such as `maxPerStageDescriptor*`, and so a descriptor set layout that is supported according to this command **must** still satisfy the pipeline layout limits such as `maxPerStageDescriptor*` in order to be used in a pipeline layout.

NOTE

This is a `VkDevice` query rather than `VkPhysicalDevice` because the answer **may** depend on enabled features.

Valid Usage (Implicit)

- VUID-vkGetDescriptorSetLayoutSupport-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkGetDescriptorSetLayoutSupport-pCreateInfo-parameter
pCreateInfo **must** be a valid pointer to a valid [VkDescriptorSetLayoutCreateInfo](#) structure
- VUID-vkGetDescriptorSetLayoutSupport-pSupport-parameter
pSupport **must** be a valid pointer to a [VkDescriptorSetLayoutSupport](#) structure

Information about support for the descriptor set layout is returned in a [VkDescriptorSetLayoutSupport](#) structure:

```
// Provided by VK_VERSION_1_1
typedef struct VkDescriptorSetLayoutSupport {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           supported;
} VkDescriptorSetLayoutSupport;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **supported** specifies whether the descriptor set layout **can** be created.

supported will be **VK_TRUE** if the descriptor set **can** be created, or else **VK_FALSE**.

Valid Usage (Implicit)

- VUID-VkDescriptorSetLayoutSupport-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_SUPPORT**
- VUID-VkDescriptorSetLayoutSupport-pNext-pNext
pNext **must** be **NULL** or a pointer to a valid instance of [VkDescriptorSetVariableDescriptorCountLayoutSupport](#)
- VUID-VkDescriptorSetLayoutSupport-sType-unique
The **sType** value of each structure in the **pNext** chain **must** be unique

If the **pNext** chain of a [VkDescriptorSetLayoutSupport](#) structure includes a [VkDescriptorSetVariableDescriptorCountLayoutSupport](#) structure, then that structure returns additional information about whether the descriptor set layout is supported.

```
// Provided by VK_VERSION_1_2
typedef struct VkDescriptorSetVariableDescriptorCountLayoutSupport {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxVariableDescriptorCount;
} VkDescriptorSetVariableDescriptorCountLayoutSupport;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxVariableDescriptorCount` indicates the maximum number of descriptors supported in the highest numbered binding of the layout, if that binding is variable-sized.

If the `VkDescriptorSetLayoutCreateInfo` structure specified in `vkGetDescriptorSetLayoutSupport::pCreateInfo` includes a variable-sized descriptor, then `supported` is determined assuming the requested size of the variable-sized descriptor, and `maxVariableDescriptorCount` is the maximum size of that descriptor that **can** be successfully created (which is greater than or equal to the requested size passed in). If the `VkDescriptorSetLayoutCreateInfo` structure does not include a variable-sized descriptor, or if the `VkPhysicalDeviceDescriptorIndexingFeatures::descriptorBindingVariableDescriptorCount` feature is not enabled, then `maxVariableDescriptorCount` is zero. For the purposes of this command, a variable-sized descriptor binding with a `descriptorCount` of zero is treated as having a `descriptorCount` of one, and thus the binding is not ignored and the maximum descriptor count will be returned. If the layout is not supported, then the value written to `maxVariableDescriptorCount` is undefined.

Valid Usage (Implicit)

- VUID-VkDescriptorSetVariableDescriptorCountLayoutSupport-sType-sType `must` be `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_LAYOUT_SUPPORT`

The following examples show a shader snippet using two descriptor sets, and application code that creates corresponding descriptor set layouts.

GLSL Example

```
//
// binding to a single sampled image descriptor in set 0
//
layout (set=0, binding=0) uniform texture2D mySampledImage;

//
// binding to an array of sampled image descriptors in set 0
//
layout (set=0, binding=1) uniform texture2D myArrayOfSampledImages[12];

//
// binding to a single uniform buffer descriptor in set 1
```

```
//
layout (set=1, binding=0) uniform myUniformBuffer
{
    vec4 myElement[32];
};
```

SPIR-V Example

```
...
%1 = OpExtInstImport "GLSL.std.450"
...
OpName %9 "mySampledImage"
OpName %14 "myArrayOfSampledImages"
OpName %18 "myUniformBuffer"
OpMemberName %18 0 "myElement"
OpName %20 ""
OpDecorate %9 DescriptorSet 0
OpDecorate %9 Binding 0
OpDecorate %14 DescriptorSet 0
OpDecorate %14 Binding 1
OpDecorate %17 ArrayStride 16
OpMemberDecorate %18 0 Offset 0
OpDecorate %18 Block
OpDecorate %20 DescriptorSet 1
OpDecorate %20 Binding 0
%2 = OpTypeVoid
%3 = OpTypeFunction %2
%6 = OpTypeFloat 32
%7 = OpTypeImage %6 2D 0 0 0 1 Unknown
%8 = OpTypePointer UniformConstant %7
%9 = OpVariable %8 UniformConstant
%10 = OpTypeInt 32 0
%11 = OpConstant %10 12
%12 = OpTypeArray %7 %11
%13 = OpTypePointer UniformConstant %12
%14 = OpVariable %13 UniformConstant
%15 = OpTypeVector %6 4
%16 = OpConstant %10 32
%17 = OpTypeArray %15 %16
%18 = OpTypeStruct %17
%19 = OpTypePointer Uniform %18
%20 = OpVariable %19 Uniform
...
```

API Example

```
VkResult myResult;

const VkDescriptorSetLayoutBinding myDescriptorSetLayoutBinding[] =
{
```

```

// binding to a single image descriptor
{
    .binding = 0,
    .descriptorType = VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE,
    .descriptorCount = 1,
    .stageFlags = VK_SHADER_STAGE_FRAGMENT_BIT,
    .pImmutableSamplers = NULL
},

// binding to an array of image descriptors
{
    .binding = 1,
    .descriptorType = VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE,
    .descriptorCount = 12,
    .stageFlags = VK_SHADER_STAGE_FRAGMENT_BIT,
    .pImmutableSamplers = NULL
},

// binding to a single uniform buffer descriptor
{
    .binding = 0,
    .descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
    .descriptorCount = 1,
    .stageFlags = VK_SHADER_STAGE_FRAGMENT_BIT,
    .pImmutableSamplers = NULL
}
};

const VkDescriptorSetLayoutCreateInfo myDescriptorSetLayoutCreateInfo[] =
{
    // Information for first descriptor set with two descriptor bindings
    {
        .sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO,
        .pNext = NULL,
        .flags = 0,
        .bindingCount = 2,
        .pBindings = &myDescriptorSetLayoutBinding[0]
    },

    // Information for second descriptor set with one descriptor binding
    {
        .sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO,
        .pNext = NULL,
        .flags = 0,
        .bindingCount = 1,
        .pBindings = &myDescriptorSetLayoutBinding[2]
    }
};

VkDescriptorSetLayout myDescriptorSetLayout[2];

```



```
//
// Create first descriptor set layout
//
myResult = vkCreateDescriptorSetLayout(
    myDevice,
    &myDescriptorSetLayoutCreateInfo[0],
    NULL,
    &myDescriptorSetLayout[0]);

//
// Create second descriptor set layout
//
myResult = vkCreateDescriptorSetLayout(
    myDevice,
    &myDescriptorSetLayoutCreateInfo[1],
    NULL,
    &myDescriptorSetLayout[1]);
```

To destroy a descriptor set layout, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyDescriptorSetLayout(
    VkDevice          device,
    VkDescriptorSetLayout descriptorSetLayout,
    const VkAllocationCallbacks* pAllocator);
```

- **device** is the logical device that destroys the descriptor set layout.
- **descriptorSetLayout** is the descriptor set layout to destroy.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage (Implicit)

- VUID-vkDestroyDescriptorSetLayout-device-parameter **device** **must** be a valid [VkDevice](#) handle
- VUID-vkDestroyDescriptorSetLayout-descriptorSetLayout-parameter
If **descriptorSetLayout** is not [VK_NULL_HANDLE](#), **descriptorSetLayout** **must** be a valid [VkDescriptorSetLayout](#) handle
- VUID-vkDestroyDescriptorSetLayout-pAllocator-null **pAllocator** **must** be [NULL](#)
- VUID-vkDestroyDescriptorSetLayout-descriptorSetLayout-parent
If **descriptorSetLayout** is a valid handle, it **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to `descriptorSetLayout` **must** be externally synchronized

15.2.2. Pipeline Layouts

Access to descriptor sets from a pipeline is accomplished through a *pipeline layout*. Zero or more descriptor set layouts and zero or more push constant ranges are combined to form a pipeline layout object describing the complete set of resources that **can** be accessed by a pipeline. The pipeline layout represents a sequence of descriptor sets with each having a specific layout. This sequence of layouts is used to determine the interface between shader stages and shader resources. Each pipeline is created using a pipeline layout.

Pipeline layout objects are represented by `VkPipelineLayout` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkPipelineLayout)
```

To create a pipeline layout, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreatePipelineLayout(
    VkDevice                                device,
    const VkPipelineLayoutCreateInfo*       pCreateInfo,
    const VkAllocationCallbacks*           pAllocator,
    VkPipelineLayout*                       pPipelineLayout);
```

- `device` is the logical device that creates the pipeline layout.
- `pCreateInfo` is a pointer to a `VkPipelineLayoutCreateInfo` structure specifying the state of the pipeline layout object.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pPipelineLayout` is a pointer to a `VkPipelineLayout` handle in which the resulting pipeline layout object is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreatePipelineLayout` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreatePipelineLayout-device-05068

The number of pipeline layouts currently allocated from `device` plus 1 **must** be less than or equal to the total number of pipeline layouts requested via `VkDeviceObjectReservationCreateInfo::pipelineLayoutRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreatePipelineLayout-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkCreatePipelineLayout-pCreateInfo-parameter
`pCreateInfo` **must** be a valid pointer to a valid `VkPipelineLayoutCreateInfo` structure
- VUID-vkCreatePipelineLayout-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreatePipelineLayout-pPipelineLayout-parameter
`pPipelineLayout` **must** be a valid pointer to a `VkPipelineLayout` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkPipelineLayoutCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineLayoutCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkPipelineLayoutCreateFlags flags;
    uint32_t           setLayoutCount;
    const VkDescriptorSetLayout* pSetLayouts;
    uint32_t           pushConstantRangeCount;
    const VkPushConstantRange* pPushConstantRanges;
} VkPipelineLayoutCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkPipelineLayoutCreateFlagBits` specifying options for pipeline layout creation.
- `setLayoutCount` is the number of descriptor sets included in the pipeline layout.
- `pSetLayouts` is a pointer to an array of `VkDescriptorSetLayout` objects. The implementation **must** not access these objects outside of the duration of the command this structure is passed to.

- `pushConstantRangeCount` is the number of push constant ranges included in the pipeline layout.
- `pPushConstantRanges` is a pointer to an array of `VkPushConstantRange` structures defining a set of push constant ranges for use in a single pipeline layout. In addition to descriptor set layouts, a pipeline layout also describes how many push constants **can** be accessed by each stage of the pipeline.

NOTE

Push constants represent a high speed path to modify constant data in pipelines that is expected to outperform memory-backed resource updates.

In Vulkan SC, the pipeline compilation process occurs `offline`, but the application **must** still provide values to `VkPipelineLayoutCreateInfo` that match the values used for offline compilation of pipelines using this `VkPipelineLayout`.

Valid Usage

- VUID-VkPipelineLayoutCreateInfo-setLayoutCount-00286
`setLayoutCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxBoundDescriptorSets`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03016
The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_SAMPLER` and `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageDescriptorSamplers`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03017
The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` and `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageDescriptorUniformBuffers`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03018
The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` and `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageDescriptorStorageBuffers`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-06939
The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, and `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER`, accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageDescriptorSampledImages`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03020
The total number of descriptors in descriptor set layouts created without the

VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, and `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageDescriptorStorageImages`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03021

The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPerStageDescriptorInputAttachments`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03022

If the `descriptorBindingSampledImageUpdateAfterBind` feature is supported on the device, the total number of descriptors with a `descriptorType` of `VK_DESCRIPTOR_TYPE_SAMPLER` and `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxPerStageDescriptorUpdateAfterBindSamplers`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03023

If the `descriptorBindingUniformBufferUpdateAfterBind` feature is supported on the device, the total number of descriptors with a `descriptorType` of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` and `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxPerStageDescriptorUpdateAfterBindUniformBuffers`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03024

If the `descriptorBindingStorageBufferUpdateAfterBind` feature is supported on the device, the total number of descriptors with a `descriptorType` of `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` and `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxPerStageDescriptorUpdateAfterBindStorageBuffers`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03025

If the `descriptorBindingSampledImageUpdateAfterBind` feature is supported on the device, the total number of descriptors with a `descriptorType` of `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, and `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxPerStageDescriptorUpdateAfterBindSampledImages`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03026

If the `descriptorBindingStorageImageUpdateAfterBind` feature is supported on the device, the total number of descriptors with a `descriptorType` of `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, and `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties`

`::maxPerStageDescriptorUpdateAfterBindStorageImages`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03027

If any element of `pSetLayouts` is created with the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set, the total number of descriptors with a `descriptorType` of `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` accessible to any given shader stage across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxPerStageDescriptorUpdateAfterBindInputAttachments`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03028

The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_SAMPLER` and `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDescriptorSetSamplers`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03029

The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDescriptorSetUniformBuffers`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03030

The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDescriptorSetUniformBuffersDynamic`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03031

The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDescriptorSetStorageBuffers`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03032

The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDescriptorSetStorageBuffersDynamic`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03033

The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, and `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDescriptorSetSampledImages`

- VUID-VkPipelineLayoutCreateInfo-descriptorType-03034
The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, and `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDescriptorSetStorageImages`
- VUID-VkPipelineLayoutCreateInfo-descriptorType-03035
The total number of descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set with a `descriptorType` of `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDescriptorSetInputAttachments`
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03036
If the `descriptorBindingSampledImageUpdateAfterBind` feature is supported on the device, the total number of descriptors of the type `VK_DESCRIPTOR_TYPE_SAMPLER` and `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxDescriptorSetUpdateAfterBindSamplers`
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03037
If the `descriptorBindingUniformBufferUpdateAfterBind` feature is supported on the device, the total number of descriptors of the type `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxDescriptorSetUpdateAfterBindUniformBuffers`
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03038
If the `descriptorBindingUniformBufferUpdateAfterBind` feature is supported on the device, the total number of descriptors of the type `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDescriptorSetUpdateAfterBindUniformBuffersDynamic`
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03039
If the `descriptorBindingStorageBufferUpdateAfterBind` feature is supported on the device, the total number of descriptors of the type `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxDescriptorSetUpdateAfterBindStorageBuffers`
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03040
If the `descriptorBindingStorageBufferUpdateAfterBind` feature is supported on the device, the total number of descriptors of the type `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDescriptorSetUpdateAfterBindStorageBuffersDynamic`
- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03041
If the `descriptorBindingSampledImageUpdateAfterBind` feature is supported on the device, the total number of descriptors of the type `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`,

`VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, and `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxDescriptorSetUpdateAfterBindSampledImages`

- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03042

If the `descriptorBindingStorageImageUpdateAfterBind` feature is supported on the device, the total number of descriptors of the type `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, and `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxDescriptorSetUpdateAfterBindStorageImages`

- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-03043

If any element of `pSetLayouts` is created with the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set, the total number of descriptors of the type `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` accessible across all shader stages and across all elements of `pSetLayouts` **must** be less than or equal to `VkPhysicalDeviceDescriptorIndexingProperties::maxDescriptorSetUpdateAfterBindInputAttachments`

- VUID-VkPipelineLayoutCreateInfo-pPushConstantRanges-00292

Any two elements of `pPushConstantRanges` **must** not include the same stage in `stageFlags`

- VUID-VkPipelineLayoutCreateInfo-graphicsPipelineLibrary-06753

Elements of `pSetLayouts` **must** be valid `VkDescriptorSetLayout` objects

Valid Usage (Implicit)

- VUID-VkPipelineLayoutCreateInfo-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO`

- VUID-VkPipelineLayoutCreateInfo-flags-zeroBitmask

`flags` **must** be 0

- VUID-VkPipelineLayoutCreateInfo-pSetLayouts-parameter

If `setLayoutCount` is not 0, `pSetLayouts` **must** be a valid pointer to an array of `setLayoutCount` valid or `VK_NULL_HANDLE` `VkDescriptorSetLayout` handles

- VUID-VkPipelineLayoutCreateInfo-pPushConstantRanges-parameter

If `pushConstantRangeCount` is not 0, `pPushConstantRanges` **must** be a valid pointer to an array of `pushConstantRangeCount` valid `VkPushConstantRange` structures

```
typedef enum VkPipelineLayoutCreateFlagBits {  
} VkPipelineLayoutCreateFlagBits;
```

All values for this enum are defined by extensions.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineLayoutCreateFlags;
```

`VkPipelineLayoutCreateFlags` is a bitmask type for setting a mask of `VkPipelineLayoutCreateFlagBits`.

The `VkPushConstantRange` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPushConstantRange {
    VkShaderStageFlags    stageFlags;
    uint32_t              offset;
    uint32_t              size;
} VkPushConstantRange;
```

- `stageFlags` is a set of stage flags describing the shader stages that will access a range of push constants. If a particular stage is not included in the range, then accessing members of that range of push constants from the corresponding shader stage will return undefined values.
- `offset` and `size` are the start offset and size, respectively, consumed by the range. Both `offset` and `size` are in units of bytes and **must** be a multiple of 4. The layout of the push constant variables is specified in the shader.

Valid Usage

- VUID-VkPushConstantRange-offset-00294
`offset` **must** be less than `VkPhysicalDeviceLimits::maxPushConstantsSize`
- VUID-VkPushConstantRange-offset-00295
`offset` **must** be a multiple of 4
- VUID-VkPushConstantRange-size-00296
`size` **must** be greater than 0
- VUID-VkPushConstantRange-size-00297
`size` **must** be a multiple of 4
- VUID-VkPushConstantRange-size-00298
`size` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPushConstantsSize` minus `offset`

Valid Usage (Implicit)

- VUID-VkPushConstantRange-stageFlags-parameter
`stageFlags` **must** be a valid combination of `VkShaderStageFlagBits` values
- VUID-VkPushConstantRange-stageFlags-requiredbitmask
`stageFlags` **must** not be 0

Once created, pipeline layouts are used as part of pipeline creation (see [Pipelines](#)), as part of binding descriptor sets (see [Descriptor Set Binding](#)), and as part of setting push constants (see [Push Constant Updates](#)). Pipeline creation accepts a pipeline layout as input, and the layout **may** be used to map (set, binding, arrayElement) tuples to implementation resources or memory locations within a descriptor set. The assignment of implementation resources depends only on the bindings defined in the descriptor sets that comprise the pipeline layout, and not on any shader source.

All resource variables [statically used](#) in all shaders in a pipeline **must** be declared with a (set, binding, arrayElement) that exists in the corresponding descriptor set layout and is of an appropriate descriptor type and includes the set of shader stages it is used by in [stageFlags](#). The pipeline layout **can** include entries that are not used by a particular pipeline. The pipeline layout allows the application to provide a consistent set of bindings across multiple pipeline compiles, which enables those pipelines to be compiled in a way that the implementation **may** cheaply switch pipelines without reprogramming the bindings.

Similarly, the push constant block declared in each shader (if present) **must** only place variables at offsets that are each included in a push constant range with [stageFlags](#) including the bit corresponding to the shader stage that uses it. The pipeline layout **can** include ranges or portions of ranges that are not used by a particular pipeline.

There is a limit on the total number of resources of each type that **can** be included in bindings in all descriptor set layouts in a pipeline layout as shown in [Pipeline Layout Resource Limits](#). The “Total Resources Available” column gives the limit on the number of each type of resource that **can** be included in bindings in all descriptor sets in the pipeline layout. Some resource types count against multiple limits. Additionally, there are limits on the total number of each type of resource that **can** be used in any pipeline stage as described in [Shader Resource Limits](#).

Table 15. Pipeline Layout Resource Limits

Total Resources Available	Resource Types
maxDescriptorSetSamplers or maxDescriptorSetUpdateAfterBindSamplers	sampler
	combined image sampler
maxDescriptorSetSampledImages or maxDescriptorSetUpdateAfterBindSampledImages	sampled image
	combined image sampler
	uniform texel buffer
maxDescriptorSetStorageImages or maxDescriptorSetUpdateAfterBindStorageImages	storage image
	storage texel buffer
maxDescriptorSetUniformBuffers or maxDescriptorSetUpdateAfterBindUniformBuffers	uniform buffer
	uniform buffer dynamic
maxDescriptorSetUniformBuffersDynamic or maxDescriptorSetUpdateAfterBindUniformBuffersDynamic	uniform buffer dynamic
maxDescriptorSetStorageBuffers or maxDescriptorSetUpdateAfterBindStorageBuffers	storage buffer
	storage buffer dynamic

Total Resources Available	Resource Types
<code>maxDescriptorSetStorageBuffersDynamic</code> or <code>maxDescriptorSetUpdateAfterBindStorageBuffersDynamic</code>	storage buffer dynamic
<code>maxDescriptorSetInputAttachments</code> or <code>maxDescriptorSetUpdateAfterBindInputAttachments</code>	input attachment

To destroy a pipeline layout, call:

```
// Provided by VK_VERSION_1_0
void vkDestroyPipelineLayout(
    VkDevice          device,
    VkPipelineLayout  pipelineLayout,
    const VkAllocationCallbacks* pAllocator);
```

- `device` is the logical device that destroys the pipeline layout.
- `pipelineLayout` is the pipeline layout to destroy.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage

- VUID-vkDestroyPipelineLayout-pipelineLayout-02004
`pipelineLayout` **must** not have been passed to any `vkCmd*` command for any command buffers that are still in the [recording state](#) when `vkDestroyPipelineLayout` is called

Valid Usage (Implicit)

- VUID-vkDestroyPipelineLayout-device-parameter
`device` **must** be a valid [VkDevice](#) handle
- VUID-vkDestroyPipelineLayout-pipelineLayout-parameter
If `pipelineLayout` is not [VK_NULL_HANDLE](#), `pipelineLayout` **must** be a valid [VkPipelineLayout](#) handle
- VUID-vkDestroyPipelineLayout-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkDestroyPipelineLayout-pipelineLayout-parent
If `pipelineLayout` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `pipelineLayout` **must** be externally synchronized

Pipeline Layout Compatibility

Two pipeline layouts are defined to be “compatible for [push constants](#)” if they were created with identical push constant ranges. Two pipeline layouts are defined to be “compatible for set N” if they were created with [identically defined](#) descriptor set layouts for sets zero through N, and if they were created with identical push constant ranges.

When binding a descriptor set (see [Descriptor Set Binding](#)) to set number N, a previously bound descriptor set bound with lower index M than N is disturbed if the pipeline layouts for set M and N are not compatible for set M. Otherwise, the bound descriptor set in M is not disturbed.

If, additionally, the previously bound descriptor set for set N was bound using a pipeline layout not compatible for set N, then all bindings in sets numbered greater than N are disturbed.

When binding a pipeline, the pipeline **can** correctly access any previously bound descriptor set N if it was bound with compatible pipeline layout for set N, and it was not disturbed.

Layout compatibility means that descriptor sets **can** be bound to a command buffer for use by any pipeline created with a compatible pipeline layout, and without having bound a particular pipeline first. It also means that descriptor sets **can** remain valid across a pipeline change, and the same resources will be accessible to the newly bound pipeline.

When a descriptor set is disturbed by binding descriptor sets, the disturbed set is considered to contain undefined descriptors bound with the same pipeline layout as the disturbing descriptor set.

Implementor’s Note

A consequence of layout compatibility is that when the implementation compiles a pipeline layout and maps pipeline resources to implementation resources, the mechanism for set N **should** only be a function of sets [0..N].

NOTE

Place the least frequently changing descriptor sets near the start of the pipeline layout, and place the descriptor sets representing the most frequently changing resources near the end. When pipelines are switched, only the descriptor set bindings that have been invalidated will need to be updated and the remainder of the descriptor set bindings will remain in place.

The maximum number of descriptor sets that **can** be bound to a pipeline layout is queried from physical device properties (see [maxBoundDescriptorSets](#) in [Limits](#)).

API Example

```
const VkDescriptorSetLayout layouts[] = { layout1, layout2 };

const VkPushConstantRange ranges[] =
{
    {
        .stageFlags = VK_SHADER_STAGE_VERTEX_BIT,
        .offset = 0,
```

```

        .size = 4
    },
    {
        .stageFlags = VK_SHADER_STAGE_FRAGMENT_BIT,
        .offset = 4,
        .size = 4
    },
};

const VkPipelineLayoutCreateInfo createInfo =
{
    .sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO,
    .pNext = NULL,
    .flags = 0,
    .setLayoutCount = 2,
    .pSetLayouts = layouts,
    .pushConstantRangeCount = 2,
    .pPushConstantRanges = ranges
};

VkPipelineLayout myPipelineLayout;
myResult = vkCreatePipelineLayout(
    myDevice,
    &createInfo,
    NULL,
    &myPipelineLayout);

```

15.2.3. Allocation of Descriptor Sets

A *descriptor pool* maintains a pool of descriptors, from which descriptor sets are allocated. Descriptor pools are externally synchronized, meaning that the application **must** not allocate and/or free descriptor sets from the same pool in multiple threads simultaneously.

Descriptor pools **cannot** be destroyed [\[SCID-4\]](#).

If `VkPhysicalDeviceVulkanSC10Properties::deviceDestroyFreesMemory` is `VK_TRUE`, the memory is returned to the system when the device is destroyed. Otherwise, it **may** not be returned to the system until the process is terminated.

Descriptor pools are represented by `VkDescriptorPool` handles:

```

// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDescriptorPool)

```

To create a descriptor pool object, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateDescriptorPool(
    VkDevice                                device,
    const VkDescriptorPoolCreateInfo*       pCreateInfo,
    const VkAllocationCallbacks*           pAllocator,
    VkDescriptorPool*                       pDescriptorPool);
```

- **device** is the logical device that creates the descriptor pool.
- **pCreateInfo** is a pointer to a [VkDescriptorPoolCreateInfo](#) structure specifying the state of the descriptor pool object.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.
- **pDescriptorPool** is a pointer to a [VkDescriptorPool](#) handle in which the resulting descriptor pool object is returned.

The created descriptor pool is returned in **pDescriptorPool**.

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is **VK_TRUE**, **vkCreateDescriptorPool** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage

- VUID-vkCreateDescriptorPool-device-05068
The number of descriptor pools currently allocated from **device** plus 1 **must** be less than or equal to the total number of descriptor pools requested via [VkDeviceObjectReservationCreateInfo::descriptorPoolRequestCount](#) specified when **device** was created

Valid Usage (Implicit)

- VUID-vkCreateDescriptorPool-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkCreateDescriptorPool-pCreateInfo-parameter
pCreateInfo **must** be a valid pointer to a valid [VkDescriptorPoolCreateInfo](#) structure
- VUID-vkCreateDescriptorPool-pAllocator-null
pAllocator **must** be **NULL**
- VUID-vkCreateDescriptorPool-pDescriptorPool-parameter
pDescriptorPool **must** be a valid pointer to a [VkDescriptorPool](#) handle
- VUID-vkCreateDescriptorPool-device-queuecount
The device **must** have been created with at least **1** queue

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

Additional information about the pool is passed in a `VkDescriptorPoolCreateInfo` structure:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorPoolCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkDescriptorPoolCreateFlags flags;
    uint32_t           maxSets;
    uint32_t           poolSizeCount;
    const VkDescriptorPoolSize* pPoolSizes;
} VkDescriptorPoolCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkDescriptorPoolCreateFlagBits` specifying certain supported operations on the pool.
- `maxSets` is the maximum number of descriptor sets that **can** be allocated from the pool.
- `poolSizeCount` is the number of elements in `pPoolSizes`.
- `pPoolSizes` is a pointer to an array of `VkDescriptorPoolSize` structures, each containing a descriptor type and number of descriptors of that type to be allocated in the pool.

If multiple `VkDescriptorPoolSize` structures containing the same descriptor type appear in the `pPoolSizes` array then the pool will be created with enough storage for the total number of descriptors of each type.

Fragmentation of a descriptor pool is possible and **may** lead to descriptor set allocation failures. A failure due to fragmentation is defined as failing a descriptor set allocation despite the sum of all outstanding descriptor set allocations from the pool plus the requested allocation requiring no more than the total number of descriptors requested at pool creation. Implementations provide certain guarantees of when fragmentation **must** not cause allocation failure, as described below.

If a descriptor pool has not had any descriptor sets freed since it was created or most recently reset then fragmentation **must** not cause an allocation failure (note that this is always the case for a pool

created without the `VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT` bit set). Additionally, if all sets allocated from the pool since it was created or most recently reset use the same number of descriptors (of each type) and the requested allocation also uses that same number of descriptors (of each type), then fragmentation **must** not cause an allocation failure.

If an allocation failure occurs due to fragmentation, an application **can** create an additional descriptor pool to perform further descriptor set allocations.

If `flags` has the `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT` bit set, descriptor pool creation **may** fail with the error `VK_ERROR_FRAGMENTATION` if the total number of descriptors across all pools (including this one) created with this bit set exceeds `maxUpdateAfterBindDescriptorsInAllPools`, or if fragmentation of the underlying hardware resources occurs.

Valid Usage

- VUID-VkDescriptorPoolCreateInfo-descriptorPoolOverallocation-09227
`maxSets` **must** be greater than 0

Valid Usage (Implicit)

- VUID-VkDescriptorPoolCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO`
- VUID-VkDescriptorPoolCreateInfo-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkDescriptorPoolCreateInfo-flags-parameter
`flags` **must** be a valid combination of `VkDescriptorPoolCreateFlagBits` values
- VUID-VkDescriptorPoolCreateInfo-pPoolSizes-parameter
If `poolSizeCount` is not 0, `pPoolSizes` **must** be a valid pointer to an array of `poolSizeCount` valid `VkDescriptorPoolSize` structures

Bits which **can** be set in `VkDescriptorPoolCreateInfo::flags`, enabling operations on a descriptor pool, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkDescriptorPoolCreateFlagBits {
    VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT = 0x00000001,
    // Provided by VK_VERSION_1_2
    VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT = 0x00000002,
} VkDescriptorPoolCreateFlagBits;
```

- `VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT` specifies that descriptor sets **can** return their individual allocations to the pool, i.e. all of `vkAllocateDescriptorSets`, `vkFreeDescriptorSets`, and `vkResetDescriptorPool` are allowed. Otherwise, descriptor sets allocated from the pool **must** not be individually freed back to the pool, i.e. only

[vkAllocateDescriptorSets](#) and [vkResetDescriptorPool](#) are allowed.

- [VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT](#) specifies that descriptor sets allocated from this pool **can** include bindings with the [VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT](#) bit set. It is valid to allocate descriptor sets that have bindings that do not set the [VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT](#) bit from a pool that has [VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT](#) set.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkDescriptorPoolCreateFlags;
```

[VkDescriptorPoolCreateFlags](#) is a bitmask type for setting a mask of zero or more [VkDescriptorPoolCreateFlagBits](#).

The [VkDescriptorPoolSize](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorPoolSize {
    VkDescriptorType    type;
    uint32_t            descriptorCount;
} VkDescriptorPoolSize;
```

- [type](#) is the type of descriptor.
- [descriptorCount](#) is the number of descriptors of that type to allocate.

NOTE

When creating a descriptor pool that will contain descriptors for combined image samplers of [multi-planar formats](#), an application needs to account for non-trivial descriptor consumption when choosing the [descriptorCount](#) value, as indicated by [VkSamplerYcbcrConversionImageFormatProperties::combinedImageSamplerDescriptorCount](#).

Valid Usage

- VUID-VkDescriptorPoolSize-descriptorCount-00302
[descriptorCount](#) **must** be greater than 0

Valid Usage (Implicit)

- VUID-VkDescriptorPoolSize-type-parameter
[type](#) **must** be a valid [VkDescriptorType](#) value

Descriptor sets are allocated from descriptor pool objects, and are represented by [VkDescriptorSet](#) handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDescriptorSet)
```

To allocate descriptor sets from a descriptor pool, call:

```
// Provided by VK_VERSION_1_0
VkResult vkAllocateDescriptorSets(
    VkDevice device,
    const VkDescriptorSetAllocateInfo* pAllocateInfo,
    VkDescriptorSet* pDescriptorSets);
```

- **device** is the logical device that owns the descriptor pool.
- **pAllocateInfo** is a pointer to a [VkDescriptorSetAllocateInfo](#) structure describing parameters of the allocation.
- **pDescriptorSets** is a pointer to an array of [VkDescriptorSet](#) handles in which the resulting descriptor set objects are returned.

The allocated descriptor sets are returned in **pDescriptorSets**.

When a descriptor set is allocated, the initial state is largely uninitialized and all descriptors are undefined, with the exception that samplers with a non-null **pImmutableSamplers** are initialized on allocation. Descriptors also become undefined if the underlying resource or view object is destroyed. Descriptor sets containing undefined descriptors **can** still be bound and used, subject to the following conditions:

- For descriptor set bindings created with the **VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT** bit set, all descriptors in that binding that are dynamically used **must** have been populated before the descriptor set is [consumed](#).
- For descriptor set bindings created without the **VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT** bit set, all descriptors in that binding that are statically used **must** have been populated before the descriptor set is [consumed](#).
- Entries that are not used by a pipeline **can** have undefined descriptors.

If a call to **vkAllocateDescriptorSets** would cause the total number of descriptor sets allocated from the pool to exceed the value of [VkDescriptorPoolCreateInfo::maxSets](#) used to create **pAllocateInfo->descriptorPool**, then the allocation **may** fail due to lack of space in the descriptor pool. Similarly, the allocation **may** fail due to lack of space if the call to **vkAllocateDescriptorSets** would cause the number of any given descriptor type to exceed the sum of all the **descriptorCount** members of each element of [VkDescriptorPoolCreateInfo::pPoolSizes](#) with a **type** equal to that type.

If the allocation fails due to no more space in the descriptor pool, and not because of system or device memory exhaustion, then **VK_ERROR_OUT_OF_POOL_MEMORY** **must** be returned.

vkAllocateDescriptorSets **can** be used to create multiple descriptor sets. If the creation of any of those descriptor sets fails, then the implementation **must** destroy all successfully created descriptor set objects from this command, set all entries of the **pDescriptorSets** array to [VK_NULL_HANDLE](#)

and return the error.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkAllocateDescriptorSets` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkAllocateDescriptorSets-device-05068

The number of descriptor sets currently allocated from `device` plus `VkDescriptorSetAllocateInfo::descriptorSetCount` **must** be less than or equal to the total number of descriptor sets requested via `VkDeviceObjectReservationCreateInfo::descriptorSetRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkAllocateDescriptorSets-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkAllocateDescriptorSets-pAllocateInfo-parameter
`pAllocateInfo` **must** be a valid pointer to a valid `VkDescriptorSetAllocateInfo` structure
- VUID-vkAllocateDescriptorSets-pDescriptorSets-parameter
`pDescriptorSets` **must** be a valid pointer to an array of `pAllocateInfo->descriptorSetCount` `VkDescriptorSet` handles
- VUID-vkAllocateDescriptorSets-device-queuecount
The device **must** have been created with at least `1` queue
- VUID-vkAllocateDescriptorSets-pAllocateInfo::descriptorSetCount-arraylength
`pAllocateInfo->descriptorSetCount` **must** be greater than `0`

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_FRAGMENTED_POOL`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_POOL_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkDescriptorSetAllocateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorSetAllocateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkDescriptorPool      descriptorPool;
    uint32_t             descriptorSetCount;
    const VkDescriptorSetLayout* pSetLayouts;
} VkDescriptorSetAllocateInfo;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **descriptorPool** is the pool which the sets will be allocated from.
- **descriptorSetCount** determines the number of descriptor sets to be allocated from the pool.
- **pSetLayouts** is a pointer to an array of descriptor set layouts, with each member specifying how the corresponding descriptor set is allocated.

Valid Usage

- VUID-VkDescriptorSetAllocateInfo-pSetLayouts-03044
If any element of **pSetLayouts** was created with the **VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT** bit set, **descriptorPool** **must** have been created with the **VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT** flag set
- VUID-VkDescriptorSetAllocateInfo-pSetLayouts-09380
If **pSetLayouts[i]** was created with an element of **pBindingFlags** that includes **VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT**, and **VkDescriptorSetVariableDescriptorCountAllocateInfo** is included in the **pNext** chain, and **VkDescriptorSetVariableDescriptorCountAllocateInfo::descriptorSetCount** is not zero, then **VkDescriptorSetVariableDescriptorCountAllocateInfo::pDescriptorCounts[i]** **must** be less than or equal to **VkDescriptorSetLayoutBinding::descriptorCount** for the corresponding binding used to create **pSetLayouts[i]**

Valid Usage (Implicit)

- VUID-VkDescriptorSetAllocateInfo-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO**
- VUID-VkDescriptorSetAllocateInfo-pNext-pNext
pNext **must** be **NULL** or a pointer to a valid instance of **VkDescriptorSetVariableDescriptorCountAllocateInfo**
- VUID-VkDescriptorSetAllocateInfo-sType-unique
The **sType** value of each structure in the **pNext** chain **must** be unique
- VUID-VkDescriptorSetAllocateInfo-descriptorPool-parameter
descriptorPool **must** be a valid **VkDescriptorPool** handle

- VUID-VkDescriptorSetAllocateInfo-pSetLayouts-parameter
`pSetLayouts` **must** be a valid pointer to an array of `descriptorSetCount` valid `VkDescriptorSetLayout` handles
- VUID-VkDescriptorSetAllocateInfo-descriptorSetCount-arraylength
`descriptorSetCount` **must** be greater than 0
- VUID-VkDescriptorSetAllocateInfo-commonparent
Both of `descriptorPool`, and the elements of `pSetLayouts` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `descriptorPool` **must** be externally synchronized

If the `pNext` chain of a `VkDescriptorSetAllocateInfo` structure includes a `VkDescriptorSetVariableDescriptorCountAllocateInfo` structure, then that structure includes an array of descriptor counts for variable-sized descriptor bindings, one for each descriptor set being allocated.

The `VkDescriptorSetVariableDescriptorCountAllocateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkDescriptorSetVariableDescriptorCountAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           descriptorSetCount;
    const uint32_t*     pDescriptorCounts;
} VkDescriptorSetVariableDescriptorCountAllocateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `descriptorSetCount` is zero or the number of elements in `pDescriptorCounts`.
- `pDescriptorCounts` is a pointer to an array of descriptor counts, with each member specifying the number of descriptors in a variable-sized descriptor binding in the corresponding descriptor set being allocated.

If `descriptorSetCount` is zero or this structure is not included in the `pNext` chain, then the variable lengths are considered to be zero. Otherwise, `pDescriptorCounts[i]` is the number of descriptors in the variable-sized descriptor binding in the corresponding descriptor set layout. If `VkDescriptorSetAllocateInfo::pSetLayouts[i]` does not include a variable-sized descriptor binding, then `pDescriptorCounts[i]` is ignored.

Valid Usage

- VUID-VkDescriptorSetVariableDescriptorCountAllocateInfo-descriptorSetCount-03045

If `descriptorSetCount` is not zero, `descriptorSetCount` **must** equal `VkDescriptorSetAllocateInfo::descriptorSetCount`

Valid Usage (Implicit)

- VUID-VkDescriptorSetVariableDescriptorCountAllocateInfo-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_ALLOCATE_INFO`
- VUID-VkDescriptorSetVariableDescriptorCountAllocateInfo-pDescriptorCounts-parameter
If `descriptorSetCount` is not 0, `pDescriptorCounts` **must** be a valid pointer to an array of `descriptorSetCount` `uint32_t` values

To free allocated descriptor sets, call:

```
// Provided by VK_VERSION_1_0
VkResult vkFreeDescriptorSets(
    VkDevice          device,
    VkDescriptorPool   descriptorPool,
    uint32_t          descriptorSetCount,
    const VkDescriptorSet* pDescriptorSets);
```

- `device` is the logical device that owns the descriptor pool.
- `descriptorPool` is the descriptor pool from which the descriptor sets were allocated.
- `descriptorSetCount` is the number of elements in the `pDescriptorSets` array.
- `pDescriptorSets` is a pointer to an array of handles to `VkDescriptorSet` objects.

After calling `vkFreeDescriptorSets`, all descriptor sets in `pDescriptorSets` are invalid.

If `recycleDescriptorSetMemory` is `VK_FALSE`, then freeing a descriptor set does not make the pool memory it used available to be reallocated until the descriptor pool is reset. If `recycleDescriptorSetMemory` is `VK_TRUE`, then the memory is available to be reallocated immediately after freeing the descriptor set.

Valid Usage

- VUID-vkFreeDescriptorSets-pDescriptorSets-00309
All submitted commands that refer to any element of `pDescriptorSets` **must** have completed execution
- VUID-vkFreeDescriptorSets-pDescriptorSets-00310
`pDescriptorSets` **must** be a valid pointer to an array of `descriptorSetCount` `VkDescriptorSet` handles, each element of which **must** either be a valid handle or `VK_NULL_HANDLE`
- VUID-vkFreeDescriptorSets-descriptorPool-00312
`descriptorPool` **must** have been created with the

VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT flag

Valid Usage (Implicit)

- VUID-vkFreeDescriptorSets-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkFreeDescriptorSets-descriptorPool-parameter
descriptorPool **must** be a valid [VkDescriptorPool](#) handle
- VUID-vkFreeDescriptorSets-descriptorSetCount-arraylength
descriptorSetCount **must** be greater than 0
- VUID-vkFreeDescriptorSets-descriptorPool-parent
descriptorPool **must** have been created, allocated, or retrieved from **device**
- VUID-vkFreeDescriptorSets-pDescriptorSets-parent
Each element of **pDescriptorSets** that is a valid handle **must** have been created, allocated, or retrieved from **descriptorPool**

Host Synchronization

- Host access to **descriptorPool** **must** be externally synchronized
- Host access to each member of **pDescriptorSets** **must** be externally synchronized

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

To return all descriptor sets allocated from a given pool to the pool, rather than freeing individual descriptor sets, call:

```
// Provided by VK_VERSION_1_0
VkResult vkResetDescriptorPool(
    VkDevice                device,
    VkDescriptorPool        descriptorPool,
    VkDescriptorPoolResetFlags flags);
```

- **device** is the logical device that owns the descriptor pool.
- **descriptorPool** is the descriptor pool to be reset.

- `flags` is reserved for future use.

Resetting a descriptor pool recycles all of the resources from all of the descriptor sets allocated from the descriptor pool back to the descriptor pool, and the descriptor sets are implicitly freed.

Valid Usage

- VUID-vkResetDescriptorPool-descriptorPool-00313
All uses of `descriptorPool` (via any allocated descriptor sets) **must** have completed execution

Valid Usage (Implicit)

- VUID-vkResetDescriptorPool-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkResetDescriptorPool-descriptorPool-parameter
`descriptorPool` **must** be a valid `VkDescriptorPool` handle
- VUID-vkResetDescriptorPool-flags-zeroBitmask
`flags` **must** be 0
- VUID-vkResetDescriptorPool-descriptorPool-parent
`descriptorPool` **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `descriptorPool` **must** be externally synchronized
- Host access to any `VkDescriptorSet` objects allocated from `descriptorPool` **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkDescriptorPoolResetFlags;
```

`VkDescriptorPoolResetFlags` is a bitmask type for setting a mask, but is currently reserved for future

use.

15.2.4. Descriptor Set Updates

Once allocated, descriptor sets **can** be updated with a combination of write and copy operations. To update descriptor sets, call:

```
// Provided by VK_VERSION_1_0
void vkUpdateDescriptorSets(
    VkDevice          device,
    uint32_t          descriptorWriteCount,
    const VkWriteDescriptorSet* pDescriptorWrites,
    uint32_t          descriptorCopyCount,
    const VkCopyDescriptorSet* pDescriptorCopies);
```

- **device** is the logical device that updates the descriptor sets.
- **descriptorWriteCount** is the number of elements in the **pDescriptorWrites** array.
- **pDescriptorWrites** is a pointer to an array of **VkWriteDescriptorSet** structures describing the descriptor sets to write to.
- **descriptorCopyCount** is the number of elements in the **pDescriptorCopies** array.
- **pDescriptorCopies** is a pointer to an array of **VkCopyDescriptorSet** structures describing the descriptor sets to copy between.

The operations described by **pDescriptorWrites** are performed first, followed by the operations described by **pDescriptorCopies**. Within each array, the operations are performed in the order they appear in the array.

Each element in the **pDescriptorWrites** array describes an operation updating the descriptor set using descriptors for resources specified in the structure.

Each element in the **pDescriptorCopies** array is a **VkCopyDescriptorSet** structure describing an operation copying descriptors between sets.

If the **dstSet** member of any element of **pDescriptorWrites** or **pDescriptorCopies** is bound, accessed, or modified by any command that was recorded to a command buffer which is currently in the **recording or executable state**, and any of the descriptor bindings that are updated were not created with the **VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT** or **VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT** bits set, that command buffer becomes **invalid**.

Copying a descriptor from a descriptor set does not constitute a use of the referenced resource or view, as it is the reference itself that is copied. Applications **can** copy a descriptor referencing a destroyed resource, and it **can** copy an undefined descriptor. The destination descriptor becomes undefined in both cases.

Valid Usage

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06236

For each element `i` where `pDescriptorWrites[i].descriptorType` is `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`, elements of the `pTexelBufferView` member of `pDescriptorWrites[i]` **must** have been created on `device`

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06237

For each element `i` where `pDescriptorWrites[i].descriptorType` is `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER`, `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER`, `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`, or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`, the `buffer` member of any element of the `pBufferInfo` member of `pDescriptorWrites[i]` **must** have been created on `device`

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06238

For each element `i` where `pDescriptorWrites[i].descriptorType` is `VK_DESCRIPTOR_TYPE_SAMPLER` or `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, and `dstSet` was not allocated with a layout that included immutable samplers for `dstBinding` with `descriptorType`, the `sampler` member of any element of the `pImageInfo` member of `pDescriptorWrites[i]` **must** have been created on `device`

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06239

For each element `i` where `pDescriptorWrites[i].descriptorType` is `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`, or `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` the `imageView` member of any element of `pDescriptorWrites[i]` **must** have been created on `device`

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06493

For each element `i` where `pDescriptorWrites[i].descriptorType` is `VK_DESCRIPTOR_TYPE_SAMPLER`, `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`, `pDescriptorWrites[i].pImageInfo` **must** be a valid pointer to an array of `pDescriptorWrites[i].descriptorCount` valid `VkDescriptorImageInfo` structures

- VUID-vkUpdateDescriptorSets-None-03047

The `dstSet` member of each element of `pDescriptorWrites` or `pDescriptorCopies` for bindings which were created without the `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` or `VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT` bits set **must** not be used by any command that was recorded to a command buffer which is in the `pending state`

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-06993

Host access to `pDescriptorWrites[i].dstSet` and `pDescriptorCopies[i].dstSet` **must** be `externally synchronized` unless explicitly denoted otherwise for specific flags

Valid Usage (Implicit)

- VUID-vkUpdateDescriptorSets-device-parameter

device **must** be a valid [VkDevice](#) handle

- VUID-vkUpdateDescriptorSets-pDescriptorWrites-parameter
If **descriptorWriteCount** is not 0, **pDescriptorWrites** **must** be a valid pointer to an array of **descriptorWriteCount** valid [VkWriteDescriptorSet](#) structures
- VUID-vkUpdateDescriptorSets-pDescriptorCopies-parameter
If **descriptorCopyCount** is not 0, **pDescriptorCopies** **must** be a valid pointer to an array of **descriptorCopyCount** valid [VkCopyDescriptorSet](#) structures

The [VkWriteDescriptorSet](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkWriteDescriptorSet {
    VkStructureType      sType;
    const void*          pNext;
    VkDescriptorSet       dstSet;
    uint32_t             dstBinding;
    uint32_t             dstArrayElement;
    uint32_t             descriptorCount;
    VkDescriptorType      descriptorType;
    const VkDescriptorImageInfo* pImageInfo;
    const VkDescriptorBufferInfo* pBufferInfo;
    const VkBufferView*    pTexelBufferView;
} VkWriteDescriptorSet;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **dstSet** is the destination descriptor set to update.
- **dstBinding** is the descriptor binding within that set.
- **dstArrayElement** is the starting element in that array.
- **descriptorCount** is the number of descriptors to update. **descriptorCount** is one of
 - the number of elements in **pImageInfo**
 - the number of elements in **pBufferInfo**
 - the number of elements in **pTexelBufferView**
- **descriptorType** is a [VkDescriptorType](#) specifying the type of each descriptor in **pImageInfo**, **pBufferInfo**, or **pTexelBufferView**, as described below. It **must** be the same type as the **descriptorType** specified in [VkDescriptorSetLayoutBinding](#) for **dstSet** at **dstBinding**. The type of the descriptor also controls which array the descriptors are taken from.
- **pImageInfo** is a pointer to an array of [VkDescriptorImageInfo](#) structures or is ignored, as described below.
- **pBufferInfo** is a pointer to an array of [VkDescriptorBufferInfo](#) structures or is ignored, as described below.
- **pTexelBufferView** is a pointer to an array of [VkBufferView](#) handles as described in the [Buffer](#)

[Views](#) section or is ignored, as described below.

Only one of `pImageInfo`, `pBufferInfo`, or `pTexelBufferView` members is used according to the descriptor type specified in the `descriptorType` member of the containing `VkWriteDescriptorSet` structure, as specified below.

If the `nullDescriptor` feature is enabled, the buffer, imageView, or bufferView **can** be `VK_NULL_HANDLE`. Loads from a null descriptor return zero values and stores and atomics to a null descriptor are discarded.

Consecutive Binding Updates

If the `dstBinding` has fewer than `descriptorCount` array elements remaining starting from `dstArrayElement`, then the remainder will be used to update the subsequent binding - `dstBinding+1` starting at array element zero. If a binding has a `descriptorCount` of zero, it is skipped. This behavior applies recursively, with the update affecting consecutive bindings as needed to update all `descriptorCount` descriptors. Consecutive bindings **must** have identical `VkDescriptorType`, `VkShaderStageFlags`, `VkDescriptorBindingFlagBits`, and immutable samplers references.

Valid Usage

- VUID-VkWriteDescriptorSet-dstBinding-00315
`dstBinding` **must** be less than or equal to the maximum value of `binding` of all `VkDescriptorSetLayoutBinding` structures specified when `dstSet`'s descriptor set layout was created
- VUID-VkWriteDescriptorSet-dstBinding-00316
`dstBinding` **must** be a binding with a non-zero `descriptorCount`
- VUID-VkWriteDescriptorSet-dstBinding-10009
`dstBinding` **must** be a binding with a non-zero `VkDescriptorSetLayoutCreateInfo::bindingCount`
- VUID-VkWriteDescriptorSet-descriptorCount-00317
All consecutive bindings updated via a single `VkWriteDescriptorSet` structure, except those with a `descriptorCount` of zero, **must** have identical `descriptorType`
- VUID-VkWriteDescriptorSet-descriptorCount-10776
All consecutive bindings updated via a single `VkWriteDescriptorSet` structure, except those with a `descriptorCount` of zero, **must** have identical `stageFlags`
- VUID-VkWriteDescriptorSet-descriptorCount-00318
All consecutive bindings updated via a single `VkWriteDescriptorSet` structure, except those with a `descriptorCount` of zero, **must** all either use immutable samplers or **must** all not use immutable samplers
- VUID-VkWriteDescriptorSet-descriptorCount-10777
All consecutive bindings updated via a single `VkWriteDescriptorSet` structure, except those with a `descriptorCount` of zero, **must** have identical `VkDescriptorBindingFlagBits`
- VUID-VkWriteDescriptorSet-descriptorType-00319
`descriptorType` **must** match the type of `dstBinding` within `dstSet`
- VUID-VkWriteDescriptorSet-dstSet-00320

`dstSet` **must** be a valid `VkDescriptorSet` handle

- VUID-VkWriteDescriptorSet-dstArrayElement-00321

The sum of `dstArrayElement` and `descriptorCount` **must** be less than or equal to the number of array elements in the descriptor set binding specified by `dstBinding`, and all applicable consecutive bindings

- VUID-VkWriteDescriptorSet-descriptorType-02994

If `descriptorType` is `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`, each element of `pTexelBufferView` **must** be either a valid `VkBufferView` handle or `VK_NULL_HANDLE`

- VUID-VkWriteDescriptorSet-descriptorType-02995

If `descriptorType` is `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` and the `nullDescriptor` feature is not enabled, each element of `pTexelBufferView` **must** not be `VK_NULL_HANDLE`

- VUID-VkWriteDescriptorSet-descriptorType-00324

If `descriptorType` is `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER`, `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER`, `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`, or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`, `pBufferInfo` **must** be a valid pointer to an array of `descriptorCount` valid `VkDescriptorBufferInfo` structures

- VUID-VkWriteDescriptorSet-descriptorType-00325

If `descriptorType` is `VK_DESCRIPTOR_TYPE_SAMPLER` or `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, and `dstSet` was not allocated with a layout that included immutable samplers for `dstBinding` with `descriptorType`, the `sampler` member of each element of `pImageInfo` **must** be a valid `VkSampler` object

- VUID-VkWriteDescriptorSet-descriptorType-02996

If `descriptorType` is `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, or `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, the `imageView` member of each element of `pImageInfo` **must** be either a valid `VkImageView` handle or `VK_NULL_HANDLE`

- VUID-VkWriteDescriptorSet-descriptorType-02997

If `descriptorType` is `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, or `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, and the `nullDescriptor` feature is not enabled, the `imageView` member of each element of `pImageInfo` **must** not be `VK_NULL_HANDLE`

- VUID-VkWriteDescriptorSet-descriptorType-07683

If `descriptorType` is `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`, the `imageView` member of each element of `pImageInfo` **must** not be `VK_NULL_HANDLE`

- VUID-VkWriteDescriptorSet-descriptorType-01946

If `descriptorType` is `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, then the `imageView` member of each `pImageInfo` element **must** have been created without a `VkSamplerYcbcrConversionInfo` structure in its `pNext` chain

- VUID-VkWriteDescriptorSet-descriptorType-02738

If `descriptorType` is `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, and if any element of `pImageInfo` has an `imageView` member that was created with a `VkSamplerYcbcrConversionInfo` structure in its `pNext` chain, then `dstSet` **must** have been

allocated with a layout that included immutable samplers for `dstBinding`, and the corresponding immutable sampler **must** have been created with an `identically defined` `VkSamplerYcbcrConversionInfo` object

- VUID-VkWriteDescriptorSet-descriptorType-01948

If `descriptorType` is `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, and `dstSet` was allocated with a layout that included immutable samplers for `dstBinding`, then the `imageView` member of each element of `pImageInfo` which corresponds to an immutable sampler that enables `sampler Y'CBCR conversion` **must** have been created with a `VkSamplerYcbcrConversionInfo` structure in its `pNext` chain with an `identically defined` `VkSamplerYcbcrConversionInfo` to the corresponding immutable sampler

- VUID-VkWriteDescriptorSet-descriptorType-09506

If `descriptorType` is `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, `dstSet` was allocated with a layout that included immutable samplers for `dstBinding`, and those samplers enable `sampler Y'CBCR conversion`, then `imageView` **must** not be `VK_NULL_HANDLE`

- VUID-VkWriteDescriptorSet-descriptorType-00327

If `descriptorType` is `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`, the `offset` member of each element of `pBufferInfo` **must** be a multiple of `VkPhysicalDeviceLimits::minUniformBufferOffsetAlignment`

- VUID-VkWriteDescriptorSet-descriptorType-00328

If `descriptorType` is `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`, the `offset` member of each element of `pBufferInfo` **must** be a multiple of `VkPhysicalDeviceLimits::minStorageBufferOffsetAlignment`

- VUID-VkWriteDescriptorSet-descriptorType-00329

If `descriptorType` is `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER`, `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`, `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER`, or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`, and the `buffer` member of any element of `pBufferInfo` is the handle of a non-sparse buffer, then that buffer **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-VkWriteDescriptorSet-descriptorType-00330

If `descriptorType` is `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`, the `buffer` member of each element of `pBufferInfo` **must** have been created with `VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT` set

- VUID-VkWriteDescriptorSet-descriptorType-00331

If `descriptorType` is `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`, the `buffer` member of each element of `pBufferInfo` **must** have been created with `VK_BUFFER_USAGE_STORAGE_BUFFER_BIT` set

- VUID-VkWriteDescriptorSet-descriptorType-00332

If `descriptorType` is `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`, the `range` member of each element of `pBufferInfo`, or the `effective range` if `range` is `VK_WHOLE_SIZE`, **must** be less than or equal to `VkPhysicalDeviceLimits::maxUniformBufferRange`

- VUID-VkWriteDescriptorSet-descriptorType-00333

If `descriptorType` is `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`, the `range` member of each element of `pBufferInfo`, or the `effective range` if `range` is `VK_WHOLE_SIZE`, **must** be less than or equal to `VkPhysicalDeviceLimits::maxStorageBufferRange`

- VUID-VkWriteDescriptorSet-descriptorType-08765

If `descriptorType` is `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER`, the `pTexelBufferView` `buffer view usage` **must** include `VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT`

- VUID-VkWriteDescriptorSet-descriptorType-08766

If `descriptorType` is `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`, the `pTexelBufferView` `buffer view usage` **must** include `VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT`

- VUID-VkWriteDescriptorSet-descriptorType-00336

If `descriptorType` is `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE` or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`, the `imageView` member of each element of `pImageInfo` **must** have been created with the `identity swizzle`

- VUID-VkWriteDescriptorSet-descriptorType-00337

If `descriptorType` is `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE` or `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, the `imageView` member of each element of `pImageInfo` **must** have been created with `VK_IMAGE_USAGE_SAMPLED_BIT` set

- VUID-VkWriteDescriptorSet-descriptorType-04149

If `descriptorType` is `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE` the `imageLayout` member of each element of `pImageInfo` **must** be a member of the list given in `Sampled Image`

- VUID-VkWriteDescriptorSet-descriptorType-04150

If `descriptorType` is `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` the `imageLayout` member of each element of `pImageInfo` **must** be a member of the list given in `Combined Image Sampler`

- VUID-VkWriteDescriptorSet-descriptorType-04151

If `descriptorType` is `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` the `imageLayout` member of each element of `pImageInfo` **must** be a member of the list given in `Input Attachment`

- VUID-VkWriteDescriptorSet-descriptorType-04152

If `descriptorType` is `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE` the `imageLayout` member of each element of `pImageInfo` **must** be a member of the list given in `Storage Image`

- VUID-VkWriteDescriptorSet-descriptorType-00338

If `descriptorType` is `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`, the `imageView` member of each element of `pImageInfo` **must** have been created with `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT` set

- VUID-VkWriteDescriptorSet-descriptorType-00339

If `descriptorType` is `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, the `imageView` member of each element of `pImageInfo` **must** have been created with `VK_IMAGE_USAGE_STORAGE_BIT` set

- VUID-VkWriteDescriptorSet-descriptorType-02752

If `descriptorType` is `VK_DESCRIPTOR_TYPE_SAMPLER`, then `dstSet` **must** not have been allocated with a layout that included immutable samplers for `dstBinding`

Valid Usage (Implicit)

- VUID-VkWriteDescriptorSet-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET`
- VUID-VkWriteDescriptorSet-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkWriteDescriptorSet-descriptorType-parameter
`descriptorType` **must** be a valid `VkDescriptorType` value
- VUID-VkWriteDescriptorSet-descriptorCount-arraylength
`descriptorCount` **must** be greater than `0`
- VUID-VkWriteDescriptorSet-commonparent
Both of `dstSet`, and the elements of `pTexelBufferView` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same `VkDevice`

The type of descriptors in a descriptor set is specified by `VkWriteDescriptorSet::descriptorType`, which **must** be one of the values:

```
// Provided by VK_VERSION_1_0
typedef enum VkDescriptorType {
    VK_DESCRIPTOR_TYPE_SAMPLER = 0,
    VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER = 1,
    VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE = 2,
    VK_DESCRIPTOR_TYPE_STORAGE_IMAGE = 3,
    VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER = 4,
    VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER = 5,
    VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER = 6,
    VK_DESCRIPTOR_TYPE_STORAGE_BUFFER = 7,
    VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC = 8,
    VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC = 9,
    VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT = 10,
} VkDescriptorType;
```

- `VK_DESCRIPTOR_TYPE_SAMPLER` specifies a `sampler descriptor`.
- `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` specifies a `combined image sampler descriptor`.
- `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE` specifies a `sampled image descriptor`.
- `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE` specifies a `storage image descriptor`.
- `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` specifies a `uniform texel buffer descriptor`.
- `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` specifies a `storage texel buffer descriptor`.
- `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` specifies a `uniform buffer descriptor`.
- `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` specifies a `storage buffer descriptor`.
- `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` specifies a `dynamic uniform buffer descriptor`.
- `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` specifies a `dynamic storage buffer descriptor`.

- `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` specifies an [input attachment descriptor](#).

When a descriptor set is updated via elements of `VkWriteDescriptorSet`, members of `pImageInfo`, `pBufferInfo` and `pTexelBufferView` are only accessed by the implementation when they correspond to descriptor type being defined - otherwise they are ignored. The members accessed are as follows for each descriptor type:

- For `VK_DESCRIPTOR_TYPE_SAMPLER`, only the `sampler` member of each element of `VkWriteDescriptorSet::pImageInfo` is accessed.
- For `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`, only the `imageView` and `imageLayout` members of each element of `VkWriteDescriptorSet::pImageInfo` are accessed.
- For `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, all members of each element of `VkWriteDescriptorSet::pImageInfo` are accessed.
- For `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER`, `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER`, `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`, or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`, all members of each element of `VkWriteDescriptorSet::pBufferInfo` are accessed.
- For `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`, each element of `VkWriteDescriptorSet::pTexelBufferView` is accessed.

The `VkDescriptorBufferInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorBufferInfo {
    VkBuffer      buffer;
    VkDeviceSize  offset;
    VkDeviceSize  range;
} VkDescriptorBufferInfo;
```

- `buffer` is `VK_NULL_HANDLE` or the buffer resource.
- `offset` is the offset in bytes from the start of `buffer`. Access to buffer memory via this descriptor uses addressing that is relative to this starting offset.
- `range` is the size in bytes that is used for this descriptor update, or `VK_WHOLE_SIZE` to use the range from `offset` to the end of the buffer.

NOTE

When setting `range` to `VK_WHOLE_SIZE`, the [effective range](#) **must** not be larger than the maximum range for the descriptor type (`maxUniformBufferRange` or `maxStorageBufferRange`). This means that `VK_WHOLE_SIZE` is not typically useful in the common case where uniform buffer descriptors are suballocated from a buffer that is much larger than `maxUniformBufferRange`.

For `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` and `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` descriptor types, `offset` is the base offset from which the dynamic offset is applied and `range` is the static size used for all dynamic offsets.

When `range` is `VK_WHOLE_SIZE` the effective range is calculated at `vkUpdateDescriptorSets` is by taking

the size of `buffer` minus the `offset`.

Valid Usage

- VUID-VkDescriptorBufferInfo-offset-00340
`offset` **must** be less than the size of `buffer`
- VUID-VkDescriptorBufferInfo-range-00341
If `range` is not equal to `VK_WHOLE_SIZE`, `range` **must** be greater than 0
- VUID-VkDescriptorBufferInfo-range-00342
If `range` is not equal to `VK_WHOLE_SIZE`, `range` **must** be less than or equal to the size of `buffer` minus `offset`
- VUID-VkDescriptorBufferInfo-buffer-02998
If the `nullDescriptor` feature is not enabled, `buffer` **must** not be `VK_NULL_HANDLE`
- VUID-VkDescriptorBufferInfo-buffer-02999
If `buffer` is `VK_NULL_HANDLE`, `offset` **must** be zero and `range` **must** be `VK_WHOLE_SIZE`

Valid Usage (Implicit)

- VUID-VkDescriptorBufferInfo-buffer-parameter
If `buffer` is not `VK_NULL_HANDLE`, `buffer` **must** be a valid `VkBuffer` handle

The `VkDescriptorImageInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDescriptorImageInfo {
    VkSampler      sampler;
    VkImageView    imageView;
    VkImageLayout  imageLayout;
} VkDescriptorImageInfo;
```

- `sampler` is a sampler handle, and is used in descriptor updates for types `VK_DESCRIPTOR_TYPE_SAMPLER` and `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` if the binding being updated does not use immutable samplers.
- `imageView` is `VK_NULL_HANDLE` or an image view handle, and is used in descriptor updates for types `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, and `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`.
- `imageLayout` is the layout that the image subresources accessible from `imageView` will be in at the time this descriptor is accessed. `imageLayout` is used in descriptor updates for types `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, and `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT`.

Members of `VkDescriptorImageInfo` that are not used in an update (as described above) are ignored.

Valid Usage

- VUID-VkDescriptorImageInfo-imageView-06712
`imageView` **must** not be a 2D array image view created from a 3D image
- VUID-VkDescriptorImageInfo-descriptorType-06713
`imageView` **must** not be a 2D view created from a 3D image
- VUID-VkDescriptorImageInfo-descriptorType-06714
`imageView` **must** not be a 2D view created from a 3D image
- VUID-VkDescriptorImageInfo-imageView-01976
If `imageView` is created from a depth/stencil image, the `aspectMask` used to create the `imageView` **must** include either `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT` but not both
- VUID-VkDescriptorImageInfo-imageLayout-09425
If `imageLayout` is `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`, then the `aspectMask` used to create `imageView` **must** not include either `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-VkDescriptorImageInfo-imageLayout-09426
If `imageLayout` is `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, then the `aspectMask` used to create `imageView` **must** not include `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkDescriptorImageInfo-sampler-01564
If `sampler` is used and the `VkFormat` of the image is a `multi-planar format`, the image **must** have been created with `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`, and the `aspectMask` of the `imageView` **must** be a valid `multi-planar aspect mask` bit

Valid Usage (Implicit)

- VUID-VkDescriptorImageInfo-commonparent
Both of `imageView`, and `sampler` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same `VkDevice`

The `VkCopyDescriptorSet` structure is defined as:


```
// Provided by VK_VERSION_1_0
typedef struct VkCopyDescriptorSet {
    VkStructureType    sType;
    const void*        pNext;
    VkDescriptorSet    srcSet;
    uint32_t           srcBinding;
    uint32_t           srcArrayElement;
    VkDescriptorSet    dstSet;
    uint32_t           dstBinding;
    uint32_t           dstArrayElement;
    uint32_t           descriptorCount;
} VkCopyDescriptorSet;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcSet`, `srcBinding`, and `srcArrayElement` are the source set, binding, and array element, respectively.
- `dstSet`, `dstBinding`, and `dstArrayElement` are the destination set, binding, and array element, respectively.
- `descriptorCount` is the number of descriptors to copy from the source to destination. If `descriptorCount` is greater than the number of remaining array elements in the source or destination binding, those affect consecutive bindings in a manner similar to `VkWriteDescriptorSet` above.

Valid Usage

- VUID-VkCopyDescriptorSet-srcBinding-00345
`srcBinding` **must** be a valid binding within `srcSet`
- VUID-VkCopyDescriptorSet-srcArrayElement-00346
The sum of `srcArrayElement` and `descriptorCount` **must** be less than or equal to the number of array elements in the descriptor set binding specified by `srcBinding`, and all applicable `consecutive bindings`
- VUID-VkCopyDescriptorSet-dstBinding-00347
`dstBinding` **must** be a valid binding within `dstSet`
- VUID-VkCopyDescriptorSet-dstArrayElement-00348
The sum of `dstArrayElement` and `descriptorCount` **must** be less than or equal to the number of array elements in the descriptor set binding specified by `dstBinding`, and all applicable `consecutive bindings`
- VUID-VkCopyDescriptorSet-dstBinding-02632
The type of `dstBinding` within `dstSet` **must** be equal to the type of `srcBinding` within `srcSet`
- VUID-VkCopyDescriptorSet-srcSet-00349
If `srcSet` is equal to `dstSet`, then the source and destination ranges of descriptors **must** not overlap, where the ranges **may** include array elements from `consecutive bindings`

- VUID-VkCopyDescriptorSet-srcSet-01918
If `srcSet`'s layout was created with the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` flag set, then `dstSet`'s layout **must** also have been created with the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` flag set
- VUID-VkCopyDescriptorSet-srcSet-04885
If `srcSet`'s layout was created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` flag set, then `dstSet`'s layout **must** have been created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` flag set
- VUID-VkCopyDescriptorSet-srcSet-01920
If the descriptor pool from which `srcSet` was allocated was created with the `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT` flag set, then the descriptor pool from which `dstSet` was allocated **must** also have been created with the `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT` flag set
- VUID-VkCopyDescriptorSet-srcSet-04887
If the descriptor pool from which `srcSet` was allocated was created without the `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT` flag set, then the descriptor pool from which `dstSet` was allocated **must** have been created without the `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT` flag set
- VUID-VkCopyDescriptorSet-dstBinding-02753
If the descriptor type of the descriptor set binding specified by `dstBinding` is `VK_DESCRIPTOR_TYPE_SAMPLER`, then `dstSet` **must** not have been allocated with a layout that included immutable samplers for `dstBinding`

Valid Usage (Implicit)

- VUID-VkCopyDescriptorSet-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_COPY_DESCRIPTOR_SET`
- VUID-VkCopyDescriptorSet-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkCopyDescriptorSet-srcSet-parameter
`srcSet` **must** be a valid `VkDescriptorSet` handle
- VUID-VkCopyDescriptorSet-dstSet-parameter
`dstSet` **must** be a valid `VkDescriptorSet` handle
- VUID-VkCopyDescriptorSet-commonparent
Both of `dstSet`, and `srcSet` **must** have been created, allocated, or retrieved from the same `VkDevice`

15.2.5. Descriptor Set Binding

To bind one or more descriptor sets to a command buffer, call:

```
// Provided by VK_VERSION_1_0
void vkCmdBindDescriptorSets(
    VkCommandBuffer                commandBuffer,
    VkPipelineBindPoint            pipelineBindPoint,
    VkPipelineLayout               layout,
    uint32_t                      firstSet,
    uint32_t                      descriptorSetCount,
    const VkDescriptorSet*         pDescriptorSets,
    uint32_t                      dynamicOffsetCount,
    const uint32_t*                pDynamicOffsets);
```

- `commandBuffer` is the command buffer that the descriptor sets will be bound to.
- `pipelineBindPoint` is a `VkPipelineBindPoint` indicating the type of the pipeline that will use the descriptors. There is a separate set of bind points for each pipeline type, so binding one does not disturb the others.
- `layout` is a `VkPipelineLayout` object used to program the bindings.
- `firstSet` is the set number of the first descriptor set to be bound.
- `descriptorSetCount` is the number of elements in the `pDescriptorSets` array.
- `pDescriptorSets` is a pointer to an array of handles to `VkDescriptorSet` objects describing the descriptor sets to bind to.
- `dynamicOffsetCount` is the number of dynamic offsets in the `pDynamicOffsets` array.
- `pDynamicOffsets` is a pointer to an array of `uint32_t` values specifying dynamic offsets.

`vkCmdBindDescriptorSets` binds descriptor sets `pDescriptorSets[0..descriptorSetCount-1]` to set numbers `[firstSet..firstSet+descriptorSetCount-1]` for subsequent `bound pipeline commands` set by `pipelineBindPoint`. Any bindings that were previously applied via these sets are no longer valid.

Once bound, a descriptor set affects rendering of subsequent commands that interact with the given pipeline type in the command buffer until either a different set is bound to the same set number, or the set is disturbed as described in [Pipeline Layout Compatibility](#).

A compatible descriptor set **must** be bound for all set numbers that any shaders in a pipeline access, at the time that a drawing or dispatching command is recorded to execute using that pipeline. However, if none of the shaders in a pipeline statically use any bindings with a particular set number, then no descriptor set need be bound for that set number, even if the pipeline layout includes a non-trivial descriptor set layout for that set number.

When consuming a descriptor, a descriptor is considered valid if the descriptor is not undefined as described by [descriptor set allocation](#). If the `nullDescriptor` feature is enabled, a null descriptor is also considered valid. A descriptor that was disturbed by [Pipeline Layout Compatibility](#), or was never bound by `vkCmdBindDescriptorSets` is not considered valid. If a pipeline accesses a descriptor either statically or dynamically depending on the `VkDescriptorBindingFlagBits`, the consuming descriptor type in the pipeline **must** match the `VkDescriptorType` in `VkDescriptorSetLayoutCreateInfo` for the descriptor to be considered valid.

NOTE

Further validation may be carried out beyond validation for descriptor types, e.g. [Texel Input Validation](#).

If any of the sets being bound include dynamic uniform or storage buffers, then `pDynamicOffsets` includes one element for each array element in each dynamic descriptor type binding in each set. Values are taken from `pDynamicOffsets` in an order such that all entries for set N come before set $N+1$; within a set, entries are ordered by the binding numbers in the descriptor set layouts; and within a binding array, elements are in order. `dynamicOffsetCount` **must** equal the total number of dynamic descriptors in the sets being bound.

The effective offset used for dynamic uniform and storage buffer bindings is the sum of the relative offset taken from `pDynamicOffsets`, and the base address of the buffer plus base offset in the descriptor set. The range of the dynamic uniform and storage buffer bindings is the buffer range as specified in the descriptor set.

Each of the `pDescriptorSets` **must** be compatible with the pipeline layout specified by `layout`. The layout used to program the bindings **must** also be compatible with the pipeline used in subsequent [bound pipeline commands](#) with that pipeline type, as defined in the [Pipeline Layout Compatibility](#) section.

The descriptor set contents bound by a call to `vkCmdBindDescriptorSets` **may** be consumed at the following times:

- For descriptor bindings created with the `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` bit set, the contents **may** be consumed when the command buffer is submitted to a queue, or during shader execution of the resulting draws and dispatches, or any time in between. Otherwise,
- during host execution of the command, or during shader execution of the resulting draws and dispatches, or any time in between.

Thus, the contents of a descriptor set binding **must** not be altered (overwritten by an update command, or freed) between the first point in time that it **may** be consumed, and when the command completes executing on the queue.

The contents of `pDynamicOffsets` are consumed immediately during execution of `vkCmdBindDescriptorSets`. Once all pending uses have completed, it is legal to update and reuse a descriptor set.

Valid Usage

- VUID-vkCmdBindDescriptorSets-pDescriptorSets-00358
Each element of `pDescriptorSets` **must** have been allocated with a `VkDescriptorSetLayout` that matches (is the same as, or identically defined as) the `VkDescriptorSetLayout` at set n in `layout`, where n is the sum of `firstSet` and the index into `pDescriptorSets`
- VUID-vkCmdBindDescriptorSets-dynamicOffsetCount-00359
`dynamicOffsetCount` **must** be equal to the total number of dynamic descriptors in `pDescriptorSets`
- VUID-vkCmdBindDescriptorSets-firstSet-00360

The sum of `firstSet` and `descriptorSetCount` **must** be less than or equal to `VkPipelineLayoutCreateInfo::setLayoutCount` provided when `layout` was created

- VUID-vkCmdBindDescriptorSets-pDynamicOffsets-01971
Each element of `pDynamicOffsets` which corresponds to a descriptor binding with type `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` **must** be a multiple of `VkPhysicalDeviceLimits::minUniformBufferOffsetAlignment`
- VUID-vkCmdBindDescriptorSets-pDynamicOffsets-01972
Each element of `pDynamicOffsets` which corresponds to a descriptor binding with type `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` **must** be a multiple of `VkPhysicalDeviceLimits::minStorageBufferOffsetAlignment`
- VUID-vkCmdBindDescriptorSets-pDescriptorSets-01979
For each dynamic uniform or storage buffer binding in `pDescriptorSets`, the sum of the `effective offset` and the range of the binding **must** be less than or equal to the size of the buffer
- VUID-vkCmdBindDescriptorSets-pDescriptorSets-06715
For each dynamic uniform or storage buffer binding in `pDescriptorSets`, if the range was set with `VK_WHOLE_SIZE` then `pDynamicOffsets` which corresponds to the descriptor binding **must** be 0
- VUID-vkCmdBindDescriptorSets-pDescriptorSets-06563
Each element of `pDescriptorSets` **must** be a valid `VkDescriptorSet`
- VUID-vkCmdBindDescriptorSets-pipelineBindPoint-00361
`pipelineBindPoint` **must** be supported by the `commandBuffer`'s parent `VkCommandPool`'s queue family

Valid Usage (Implicit)

- VUID-vkCmdBindDescriptorSets-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdBindDescriptorSets-pipelineBindPoint-parameter
`pipelineBindPoint` **must** be a valid `VkPipelineBindPoint` value
- VUID-vkCmdBindDescriptorSets-layout-parameter
`layout` **must** be a valid `VkPipelineLayout` handle
- VUID-vkCmdBindDescriptorSets-pDescriptorSets-parameter
`pDescriptorSets` **must** be a valid pointer to an array of `descriptorSetCount` valid or `VK_NULL_HANDLE` `VkDescriptorSet` handles
- VUID-vkCmdBindDescriptorSets-pDynamicOffsets-parameter
If `dynamicOffsetCount` is not 0, `pDynamicOffsets` **must** be a valid pointer to an array of `dynamicOffsetCount` `uint32_t` values
- VUID-vkCmdBindDescriptorSets-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdBindDescriptorSets-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support

VK_QUEUE_COMPUTE_BIT, or VK_QUEUE_GRAPHICS_BIT operations

- VUID-vkCmdBindDescriptorSets-descriptorSetCount-arraylength
descriptorSetCount **must** be greater than 0

- VUID-vkCmdBindDescriptorSets-commonparent

Each of commandBuffer, layout, and the elements of pDescriptorSets that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same VkDevice

Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT	State

15.2.6. Push Constant Updates

As described above in section [Pipeline Layouts](#), the pipeline layout defines shader push constants which are updated via Vulkan commands rather than via writes to memory or copy commands.

NOTE

Push constants represent a high speed path to modify constant data in pipelines that is expected to outperform memory-backed resource updates.

To update push constants, call:

```
// Provided by VK_VERSION_1_0
void vkCmdPushConstants(
    VkCommandBuffer          commandBuffer,
    VkPipelineLayout         layout,
    VkShaderStageFlags       stageFlags,
    uint32_t                 offset,
    uint32_t                 size,
    const void*              pValues);
```

- `commandBuffer` is the command buffer in which the push constant update will be recorded.
- `layout` is the pipeline layout used to program the push constant updates.
- `stageFlags` is a bitmask of `VkShaderStageFlagBits` specifying the shader stages that will use the push constants in the updated range.
- `offset` is the start offset of the push constant range to update, in units of bytes.
- `size` is the size of the push constant range to update, in units of bytes.
- `pValues` is a pointer to an array of `size` bytes containing the new push constant values.

When a command buffer begins recording, all push constant values are undefined.

Push constant values **can** be updated incrementally, causing shader stages in `stageFlags` to read the new data from `pValues` for push constants modified by this command, while still reading the previous data for push constants not modified by this command. When a `bind pipeline command` is issued, the bound pipeline's layout **must** be compatible with the layouts used to set the values of all push constants in the pipeline layout's push constant ranges, as described in [Pipeline Layout Compatibility](#). Binding a pipeline with a layout that is not compatible with the push constant layout does not disturb the push constant values.

NOTE

As `stageFlags` needs to include all flags the relevant push constant ranges were created with, any flags that are not supported by the queue family that the `VkCommandPool` used to allocate `commandBuffer` was created on are ignored.

Valid Usage

- VUID-vkCmdPushConstants-offset-01795
For each byte in the range specified by `offset` and `size` and for each shader stage in `stageFlags`, there **must** be a push constant range in `layout` that includes that byte and that stage
- VUID-vkCmdPushConstants-offset-01796
For each byte in the range specified by `offset` and `size` and for each push constant range that overlaps that byte, `stageFlags` **must** include all stages in that push constant range's `VkPushConstantRange::stageFlags`
- VUID-vkCmdPushConstants-offset-00368
`offset` **must** be a multiple of 4
- VUID-vkCmdPushConstants-size-00369
`size` **must** be a multiple of 4
- VUID-vkCmdPushConstants-offset-00370
`offset` **must** be less than `VkPhysicalDeviceLimits::maxPushConstantsSize`
- VUID-vkCmdPushConstants-size-00371
`size` **must** be less than or equal to `VkPhysicalDeviceLimits::maxPushConstantsSize` minus `offset`

Valid Usage (Implicit)

- VUID-vkCmdPushConstants-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdPushConstants-layout-parameter
`layout` **must** be a valid `VkPipelineLayout` handle
- VUID-vkCmdPushConstants-stageFlags-parameter
`stageFlags` **must** be a valid combination of `VkShaderStageFlagBits` values
- VUID-vkCmdPushConstants-stageFlags-requiredbitmask
`stageFlags` **must** not be `0`
- VUID-vkCmdPushConstants-pValues-parameter
`pValues` **must** be a valid pointer to an array of `size` bytes
- VUID-vkCmdPushConstants-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdPushConstants-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdPushConstants-size-arraylength
`size` **must** be greater than `0`
- VUID-vkCmdPushConstants-commonparent
Both of `commandBuffer`, and `layout` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT	State

15.3. Physical Storage Buffer Access

Buffer device addresses **can** also be used to access buffer memory in a shader, using the `SPV_KHR_physical_storage_buffer` extension and the `PhysicalStorageBuffer` storage class. For example, this value **can** be stored in a uniform buffer, and the shader **can** read the value from the uniform buffer and use it to do a dependent read/write to this buffer. All loads, stores, and atomics in a shader through `PhysicalStorageBuffer` pointers **must** access addresses in the address range of some buffer.

Chapter 16. Shader Interfaces

When a pipeline is created, the set of shaders specified in the corresponding `VkPipelineCreateInfo` structure are implicitly linked at a number of different interfaces.

- [Shader Input and Output Interface](#)
- [Vertex Input Interface](#)
- [Fragment Output Interface](#)
- [Fragment Input Attachment Interface](#)
- [Shader Resource Interface](#)

In Vulkan SC, the pipeline compilation process occurs [offline](#) using the implementation-provided pipeline cache compiler. The set of shaders being used to create a pipeline **can** be specified using the pipeline JSON schema.

This chapter describes valid uses for a set of SPIR-V decorations. Any other use of one of these decorations is invalid, with the exception that, when using SPIR-V versions 1.4 and earlier: `Block`, `BufferBlock`, `Offset`, `ArrayStride`, and `MatrixStride` can also decorate types and type members used by variables in the `Private` and `Function` storage classes.

NOTE

In this chapter, there are references to SPIR-V terms such as the `MeshNV` execution model. These terms will appear even in a build of the specification which does not support any extensions. This is as intended, since these terms appear in the unified SPIR-V specification without such qualifiers.

16.1. Shader Input and Output Interfaces

When multiple stages are present in a pipeline, the outputs of one stage form an interface with the inputs of the next stage. When such an interface involves a shader, shader outputs are matched against the inputs of the next stage, and shader inputs are matched against the outputs of the previous stage.

All the variables forming the shader input and output *interfaces* are listed as operands to the `OpEntryPoint` instruction and are declared with the `Input` or `Output` storage classes, respectively, in the SPIR-V module. These generally form the interfaces between consecutive shader stages, regardless of any non-shader stages between the consecutive shader stages.

There are two classes of variables that **can** be matched between shader stages, built-in variables and user-defined variables. Each class has a different set of matching criteria.

`Output` variables of a shader stage have undefined values until the shader writes to them or uses the `Initializer` operand when declaring the variable.

For compute shaders, the input interface is formed by the built-in interface. The output interface is empty.

16.1.1. Built-In Interface Block

Shader **built-in** variables meeting the following requirements define the *built-in interface block*. They **must**

- be explicitly declared (there are no implicit built-ins),
- be identified with a **BuiltIn** decoration,
- form object types as described in the **Built-in Variables** section, and
- be declared in a block whose top-level members are the built-ins.

There **must** be no more than one built-in interface block per shader per interface .

Built-ins **must** not have any **Location** or **Component** decorations.

16.1.2. User-Defined Variable Interface

The non-built-in variables listed by **OpEntryPoint** with the **Input** or **Output** storage class form the *user-defined variable interface*. These **must** have **numeric type** or, recursively, composite types of such types. If an implementation supports **storageInputOutput16**, components **can** have a width of 16 bits. These variables **must** be identified with a **Location** decoration and **can** also be identified with a **Component** decoration.

16.1.3. Interface Matching

An output variable, block, or structure member in a given shader stage has an interface match with an input variable, block, or structure member in a subsequent shader stage if they both adhere to the following conditions:

- They have equivalent decorations, other than:
 - one is not decorated with **Component** and the other is declared with a **Component** of 0
 - **Interpolation decorations**
 - **RelaxedPrecision** if one is an input variable and the other an output variable
- Their types match as follows:
 - if the input is declared in a tessellation control or geometry shader as an **OpTypeArray** with an **Element Type** equivalent to the **OpType*** declaration of the output, and neither is a structure member; or
 - if in any other case they are declared with an equivalent **OpType*** declaration.
- If both are structures and every member has an interface match.

NOTE

The word “structure” above refers to both variables that have an **OpTypeStruct** type and interface blocks (which are also declared as **OpTypeStruct**).

All input variables and blocks **must** have an interface match in the preceding shader stage, except for built-in variables in fragment shaders. Shaders **can** declare and write to output variables that are not declared or read by the subsequent stage.

The value of an input variable is undefined if the preceding stage does not write to a matching output variable, as described above.

16.1.4. Location and Component Assignment

User-defined variables in interfaces between shader stages in the graphics pipeline consume a unique set of Location and Component values.

Available space for user-defined interface variables is partitioned into a number of 32-bit four-component vectors, each identified by a Location value. Each individual 32-bit component of a vector is then further identified by a Component value.

16-bit scalar or vector values consume one Component slot per 16-bit component and **must** be specified within a single Location. 32-bit scalar or vector values consume one Component slot per 32-bit component and **must** be specified within a single Location. 64-bit scalar or vector values consume two consecutive Component slots per 64-bit component from up to two consecutive Location slots.

For any shader interface variable where one level of the array is disregarded for type matching, the outer array level is also disregarded when assigning Location slots.

An array of size n with elements consuming l Location slots each will consume $l \times n$ Location slots. Each element of the array will consume Component slots in each Location slot identically to a declaration using the element type.

Matrices of size $n \times m$ are assigned locations identically to arrays of size n of vectors of length 4 (consuming all Component slots) with an identical element type.

When a variable with a structure type is decorated with a Location, the members in the structure type **must** not be decorated with a Location. The variable's members are assigned consecutive locations in declaration order, starting from the first member, which is assigned the location decoration from the variable. The Location slots consumed by structure members are determined by applying the rules above in a depth-first traversal of the instantiated members as though the structure or block member were declared as an input or output variable of the same type.

A variable with a structure type that is not decorated with Block **must** be decorated with a Location.

When a variable with a structure type decorated with Block is declared without a Location decoration, each member in the structure **must** be decorated with a Location. Types nested deeper than the top-level members **must** not have Location decorations.

Multiple variable declarations in the same storage class **must** not have overlapping Component slots within the same Location.

The number of input and output locations available for a shader input or output interface depend on the shader stage as described in Shader Input and Output Locations. All variables in both the built-in interface block and the user-defined variable interface count against these limits. Each effective Location **must** have a value less than the number of Location slots available for the given interface, as specified in the "Locations Available" column in Shader Input and Output Locations.

Table 16. Shader Input and Output Locations

Shader Interface	Locations Available
vertex input	<code>maxVertexInputAttributes</code>
vertex output	<code>maxVertexOutputComponents</code> / 4
tessellation control input	<code>maxTessellationControlPerVertexInputComponents</code> / 4
tessellation control output	<code>maxTessellationControlPerVertexOutputComponents</code> / 4
tessellation evaluation input	<code>maxTessellationEvaluationInputComponents</code> / 4
tessellation evaluation output	<code>maxTessellationEvaluationOutputComponents</code> / 4
geometry input	<code>maxGeometryInputComponents</code> / 4
geometry output	<code>maxGeometryOutputComponents</code> / 4
fragment input	<code>maxFragmentInputComponents</code> / 4
fragment output	<code>maxFragmentOutputAttachments</code>

16.2. Vertex Input Interface

When the vertex stage is present in a pipeline, the vertex shader input variables form an interface with the vertex input attributes. The vertex shader input variables are matched by the `Location` and `Component` decorations to the vertex input attributes specified in the `pVertexInputState` member of the `VkGraphicsPipelineCreateInfo` structure.

The vertex shader input variables listed by `OpEntryPoint` with the `Input` storage class form the *vertex input interface*. These variables **must** be identified with a `Location` decoration and **can** also be identified with a `Component` decoration.

For the purposes of interface matching: variables declared without a `Component` decoration are considered to have a `Component` decoration of zero. The number of available vertex input `Location` slots is given by the `maxVertexInputAttributes` member of the `VkPhysicalDeviceLimits` structure.

See [Attribute Location and Component Assignment](#) for details.

All vertex shader inputs declared as above **must** have a corresponding attribute and binding in the pipeline.

Components and locations are consumed as defined for [Location and Component Assignment](#). Multiple [user-defined input variable](#) declarations **must** not have overlapping `Component` slots within the same `Location`.

16.3. Fragment Output Interface

When the fragment stage is present in a pipeline, the fragment shader outputs form an interface with the output attachments defined by a [render pass instance](#). The fragment shader output variables are matched by the `Location` and `Component` decorations to specified color attachments.

The fragment shader output variables listed by `OpEntryPoint` with the `Output` storage class form the *fragment output interface*. These variables **must** be identified with a `Location` decoration. They **can** also be identified with a `Component` decoration and/or an `Index` decoration. For the purposes of interface matching: variables declared without a `Component` decoration are considered to have a `Component` decoration of zero, and variables declared without an `Index` decoration are considered to have an `Index` decoration of zero.

A fragment shader output variable identified with a `Location` decoration of i is associated with the color attachment indicated by `VkSubpassDescription::pColorAttachments[i]`. Values are written to those attachments after passing through the blending unit as described in [Blending](#), if enabled. The number of available fragment output `Location` slots is given by the `maxFragmentOutputAttachments` member of the `VkPhysicalDeviceLimits` structure.

When an active fragment shader invocation finishes, the values of all fragment shader outputs are copied out and used as blend inputs or color attachments writes. If the invocation does not set a value for them, the input values to those blending or color attachment writes are undefined. If there is no color attachment indicated by `Location`, the values that would have been written to the color attachments are discarded.

Output `Component` words identified as 0, 1, 2, and 3 will be directed to the R, G, B, and A inputs to the blending unit, respectively, or to the output attachment if blending is disabled. If two variables are placed within the same `Location`, they **must** have the same underlying type (floating-point or integer). `Component` words which do not correspond to any fragment shader output will also result in undefined values for blending or color attachment writes.

Fragment outputs identified with an `Index` of zero are directed to the first input of the blending unit associated with the corresponding `Location`. Outputs identified with an `Index` of one are directed to the second input of the corresponding blending unit.

Components and locations are consumed as defined for [Location and Component Assignment](#). Output variable declarations **must** not consume any of the same `Component` slots within the same `Location` and with the same `Index` value as any other output variable declaration.

Output values written by a fragment shader **must** be declared with either `OpTypeFloat` or `OpTypeInt`, and a `Width` of 32. If `storageInputOutput16` is supported, output values written by a fragment shader **can** be also declared with either `OpTypeFloat` or `OpTypeInt` and a `Width` of 16. Composites of these types are also permitted. If the color attachment has a signed or unsigned normalized fixed-point format, color values are assumed to be floating-point and are converted to fixed-point as described in [Conversion From Floating-Point to Normalized Fixed-Point](#); If the color attachment has an integer format, color values are assumed to be integers and converted to the bit-depth of the target. Any value that cannot be represented in the attachment's format is undefined. For any other attachment format no conversion is performed. If the type of the values written by the fragment shader do not match the format of the corresponding color attachment, the resulting values are undefined for those components.

16.4. Fragment Input Attachment Interface

When a fragment stage is present in a pipeline, the fragment shader subpass inputs form an interface with the input attachments of the current subpass. The fragment shader subpass input

variables are matched by `InputAttachmentIndex` decorations to the input attachments specified in the `pInputAttachments` array of the `VkSubpassDescription` structure describing the subpass that the fragment shader is executed in.

The fragment shader subpass input variables with the `UniformConstant` storage class and a decoration of `InputAttachmentIndex` that are statically used by `OpEntryPoint` form the *fragment input attachment interface*. These variables **must** be declared with a type of `OpTypeImage`, a `Dim` operand of `SubpassData`, an `Arrayed` operand of 0, and a `Sampled` operand of 2. The `MS` operand of the `OpTypeImage` **must** be 0 if the `samples` field of the corresponding `VkAttachmentDescription` is `VK_SAMPLE_COUNT_1_BIT` and 1 otherwise.

A subpass input variable identified with an `InputAttachmentIndex` decoration of *i* reads from the input attachment indicated by `pInputAttachments[i]` member of `VkSubpassDescription`. If the subpass input variable is declared as an array of size *N*, it consumes *N* consecutive input attachments, starting with the index specified. There **must** not be more than one input variable with the same `InputAttachmentIndex` whether explicitly declared or implied by an array declaration per image aspect. A multi-aspect image (e.g. a depth/stencil format) **can** use the same input variable. The number of available input attachment indices is given by the `maxPerStageDescriptorInputAttachments` member of the `VkPhysicalDeviceLimits` structure.

Variables identified with the `InputAttachmentIndex` **must** only be used by a fragment stage. The `numeric format` of the subpass input **must** match the format of the corresponding input attachment, or the values of subpass loads from these variables are undefined. If the framebuffer attachment contains both depth and stencil aspects, the numeric format of the subpass input determines if depth or stencil aspect is accessed by the shader.

See [Input Attachment](#) for more details.

16.4.1. Fragment Input Attachment Compatibility

An input attachment that is statically accessed by a fragment shader **must** be backed by a descriptor that is equivalent to the `VkImageView` in the `VkFramebuffer`, except for `subresourceRange.aspectMask`. The `aspectMask` **must** be equal to the aspect accessed by the shader.

16.5. Shader Resource Interface

When a shader stage accesses buffer or image resources, as described in the [Resource Descriptors](#) section, the shader resource variables **must** be matched with the [pipeline layout](#) that is provided at pipeline creation time.

The set of shader variables that form the *shader resource interface* for a stage are the variables statically used by that stage's `OpEntryPoint` with a storage class of `Uniform`, `UniformConstant`, `StorageBuffer`, or `PushConstant`. For the fragment shader, this includes the [fragment input attachment interface](#).

The shader resource interface consists of two sub-interfaces: the push constant interface and the descriptor set interface.

16.5.1. Push Constant Interface

The shader variables defined with a storage class of **PushConstant** that are statically used by the shader entry points for the pipeline define the *push constant interface*. They **must** be:

- typed as **OpTypeStruct**,
- identified with a **Block** decoration, and
- laid out explicitly using the **Offset**, **ArrayStride**, and **MatrixStride** decorations as specified in [Offset and Stride Assignment](#).

There **must** be no more than one push constant block statically used per shader entry point.

Each statically used member of a push constant block **must** be placed at an **Offset** such that the entire member is entirely contained within the **VkPushConstantRange** for each **OpEntryPoint** that uses it, and the **stageFlags** for that range **must** specify the appropriate **VkShaderStageFlagBits** for that stage. The **Offset** decoration for any member of a push constant block **must** not cause the space required for that member to extend outside the range [0, **maxPushConstantsSize**).

Any member of a push constant block that is declared as an array **must** only be accessed with *dynamically uniform* indices.

16.5.2. Descriptor Set Interface

The *descriptor set interface* is comprised of the shader variables with the storage class of **StorageBuffer**, **Uniform** or **UniformConstant** (including the variables in the [fragment input attachment interface](#)) that are statically used by the shader entry points for the pipeline.

These variables **must** have **DescriptorSet** and **Binding** decorations specified, which are assigned and matched with the **VkDescriptorSetLayout** objects in the pipeline layout as described in [DescriptorSet and Binding Assignment](#).

The **Image Format** of an **OpTypeImage** declaration **must** not be **Unknown**, for variables which are used for **OpImageRead**, **OpImageSparseRead**, or **OpImageWrite** operations, except under the following conditions:

- For **OpImageWrite**, if the image format is listed in the [storage without format](#) list and if the **shaderStorageImageWriteWithoutFormat** feature is enabled and the shader module declares the **StorageImageWriteWithoutFormat** capability.
- For **OpImageRead** or **OpImageSparseRead**, if the image format is listed in the [storage without format](#) list and if the **shaderStorageImageReadWithoutFormat** feature is enabled and the shader module declares the **StorageImageReadWithoutFormat** capability.
- For **OpImageRead**, if **Dim** is **SubpassData** (indicating a read from an input attachment).

The **Image Format** of an **OpTypeImage** declaration **must** not be **Unknown**, for variables which are used for **OpAtomic*** operations.

Variables identified with the **Uniform** storage class are used to access transparent buffer backed resources. Such variables **must** be:

- typed as `OpTypeStruct`, or an array of this type,
- identified with a `Block` or `BufferBlock` decoration, and
- laid out explicitly using the `Offset`, `ArrayStride`, and `MatrixStride` decorations as specified in [Offset and Stride Assignment](#).

Variables identified with the `StorageBuffer` storage class are used to access transparent buffer backed resources. Such variables **must** be:

- typed as `OpTypeStruct`, or an array of this type,
- identified with a `Block` decoration, and
- laid out explicitly using the `Offset`, `ArrayStride`, and `MatrixStride` decorations as specified in [Offset and Stride Assignment](#).

The `Offset` decoration for any member of a `Block`-decorated variable in the `Uniform` storage class **must** not cause the space required for that variable to extend outside the range `[0, maxUniformBufferRange)`. The `Offset` decoration for any member of a `Block`-decorated variable in the `StorageBuffer` storage class **must** not cause the space required for that variable to extend outside the range `[0, maxStorageBufferRange)`.

Variables identified with a storage class of `UniformConstant` and a decoration of `InputAttachmentIndex` **must** be declared as described in [Fragment Input Attachment Interface](#).

SPIR-V variables decorated with a descriptor set and binding that identify a [combined image sampler descriptor](#) **can** have a type of `OpTypeImage`, `OpTypeSampler` (`Sampled=1`), or `OpTypeSampledImage`.

When accessing a resource through such a variable, the resource **must** be selected via compile time constant expressions unless features are enabled to allow dynamically uniform or non-uniform expressions, as described below:

- Storage images (except storage texel buffers and input attachments):
 - Dynamically uniform: `shaderStorageImageArrayDynamicIndexing` and `StorageImageArrayDynamicIndexing`
 - Non-uniform: `shaderStorageImageArrayNonUniformIndexing` and `StorageImageArrayNonUniformIndexing`
- Storage texel buffers:
 - Dynamically uniform: `shaderStorageTexelBufferArrayDynamicIndexing` and `StorageTexelBufferArrayDynamicIndexing`
 - Non-uniform: `shaderStorageTexelBufferArrayNonUniformIndexing` and `StorageTexelBufferArrayNonUniformIndexing`
- Input attachments:
 - Dynamically uniform: `shaderInputAttachmentArrayDynamicIndexing` and `InputAttachmentArrayDynamicIndexing`
 - Non-uniform: `shaderInputAttachmentArrayNonUniformIndexing` and `InputAttachmentArrayNonUniformIndexing`

- Sampled images (except uniform texel buffers), samplers and combined image samplers:
 - Dynamically uniform: `shaderSampledImageArrayDynamicIndexing` and `SampledImageArrayDynamicIndexing`
 - Non-uniform: `shaderSampledImageArrayNonUniformIndexing` and `SampledImageArrayNonUniformIndexing`
- Uniform texel buffers:
 - Dynamically uniform: `shaderUniformTexelBufferArrayDynamicIndexing` and `UniformTexelBufferArrayDynamicIndexing`
 - Non-uniform: `shaderUniformTexelBufferArrayNonUniformIndexing` and `UniformTexelBufferArrayNonUniformIndexing`
- Uniform buffers:
 - Dynamically uniform: `shaderUniformBufferArrayDynamicIndexing` and `UniformBufferArrayDynamicIndexing`
 - Non-uniform: `shaderUniformBufferArrayNonUniformIndexing` and `UniformBufferArrayNonUniformIndexing`
- Storage buffers:
 - Dynamically uniform: `shaderStorageBufferArrayDynamicIndexing` and `StorageBufferArrayDynamicIndexing`
 - Non-uniform: `shaderStorageBufferArrayNonUniformIndexing` and `StorageBufferArrayNonUniformIndexing`

NOTE

Implementations must take care when implementing this if subgroups are not necessarily a subset of the invocation group for their hardware (e.g. if multiple draw calls are packed together). If such an implementation needs uniformity across the subgroup for any resource access, the implementation is responsible for ensuring that indices that are only dynamically uniform across the invocation group still function as expected if they advertise the respective feature.

A combined image sampler in an array that enables sampler $Y'C_B C_R$ conversion **must** only be indexed by constant integral expressions.

Table 17. Shader Resource and Descriptor Type Correspondence

Resource type	Descriptor Type
sampler	<code>VK_DESCRIPTOR_TYPE_SAMPLER</code> or <code>VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER</code>
sampled image	<code>VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE</code> or <code>VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER</code>
storage image	<code>VK_DESCRIPTOR_TYPE_STORAGE_IMAGE</code>
combined image sampler	<code>VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER</code>
uniform texel buffer	<code>VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER</code>
storage texel buffer	<code>VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER</code>

Resource type	Descriptor Type
uniform buffer	VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC
storage buffer	VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC
input attachment	VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT

Table 18. Shader Resource and Storage Class Correspondence

Resource type	Storage Class	Type ¹	Decoration(s) ²
sampler	UniformConstant	OpTypeSampler	
sampled image	UniformConstant	OpTypeImage (Sampled=1)	
storage image	UniformConstant	OpTypeImage (Sampled=2)	
combined image sampler	UniformConstant	OpTypeSampledImage OpTypeImage (Sampled=1) OpTypeSampler	
uniform texel buffer	UniformConstant	OpTypeImage (Dim=Buffer, Sampled=1)	
storage texel buffer	UniformConstant	OpTypeImage (Dim=Buffer, Sampled=2)	
uniform buffer	Uniform	OpTypeStruct	Block, Offset, (ArrayStride), (MatrixStride)
storage buffer	Uniform	OpTypeStruct	BufferBlock, Offset, (ArrayStride), (MatrixStride)
	StorageBuffer		Block, Offset, (ArrayStride), (MatrixStride)
input attachment	UniformConstant	OpTypeImage (Dim=SubpassData, Sampled=2)	InputAttachmentIndex

1

Where **OpTypeImage** is referenced, the **Dim** values **Buffer** and **Subpassdata** are only accepted where they are specifically referenced. They do not correspond to resource types where a generic **OpTypeImage** is specified.

2

In addition to **DescriptorSet** and **Binding**.

16.5.3. DescriptorSet and Binding Assignment

A variable decorated with a **DescriptorSet** decoration of **s** and a **Binding** decoration of **b** indicates that this variable is associated with the **VkDescriptorSetLayoutBinding** that has a **binding** equal to **b** in **pSetLayouts[s]** that was specified in **VkPipelineLayoutCreateInfo**.

DescriptorSet decoration values **must** be between zero and **maxBoundDescriptorSets** minus one,

inclusive. **Binding** decoration values **can** be any 32-bit unsigned integer value, as described in [Descriptor Set Layout](#). Each descriptor set has its own binding name space.

If the **Binding** decoration is used with an array, the entire array is assigned that binding value. The decorated array **must** have an **Element Type** corresponding to a descriptor type, and the size of the array **must** be no larger than the number of descriptors in the binding. If the array is runtime-sized, then array elements greater than or equal to the size of that binding in the bound descriptor set **must** not be used. If the array is runtime-sized, the **runtimeDescriptorArray** feature **must** be enabled and the **RuntimeDescriptorArray** capability **must** be declared. The index of each element of the array is referred to as the *arrayElement*. For the purposes of interface matching and descriptor set [operations](#), if a resource variable is not an array, it is treated as if it has an *arrayElement* of zero.

There is a limit on the number of resources of each type that **can** be accessed by a pipeline stage as shown in [Shader Resource Limits](#). The “Resources Per Stage” column gives the limit on the number each type of resource that **can** be statically used for an entry point in any given stage in a pipeline. The “Resource Types” column lists which resource types are counted against the limit. Some resource types count against multiple limits.

The pipeline layout **may** include descriptor sets and bindings which are not referenced by any variables statically used by the entry points for the shader stages in the binding’s **stageFlags**.

However, if a variable assigned to a given **DescriptorSet** and **Binding** is statically used by the entry point for a shader stage, the pipeline layout **must** contain a descriptor set layout binding in that descriptor set layout and for that binding number, and that binding’s **stageFlags** **must** include the appropriate **VkShaderStageFlagBits** for that stage. The variable **must** be of a valid resource type determined by its SPIR-V type and storage class, as defined in [Shader Resource and Storage Class Correspondence](#). The descriptor set layout binding **must** be of a corresponding descriptor type, as defined in [Shader Resource and Descriptor Type Correspondence](#).

NOTE

There are no limits on the number of shader variables that can have overlapping set and binding values in a shader; but which resources are [statically used](#) has an impact. If any shader variable identifying a resource is [statically used](#) in a shader, then the underlying descriptor bound at the declared set and binding **must support the declared type in the shader** when the shader executes.

If multiple shader variables are declared with the same set and binding values, and with the same underlying descriptor type, they can all be statically used within the same shader. However, accesses are not automatically synchronized, and **Aliased** decorations should be used to avoid data hazards (see [section 2.18.2 Aliasing in the SPIR-V specification](#)).

If multiple shader variables with the same set and binding values are declared in a single shader, but with different declared types, where any of those are not supported by the relevant bound descriptor, that shader can only be executed if the variables with the unsupported type are not statically used.

A noteworthy example of using multiple statically-used shader variables sharing the same descriptor set and binding values is a descriptor of type **VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER** that has multiple corresponding shader

variables in the `UniformConstant` storage class, where some could be `OpTypeImage (Sampled=1)`, some could be `OpTypeSampler`, and some could be `OpTypeSampledImage`.

Table 19. Shader Resource Limits

Resources per Stage	Resource Types
<code>maxPerStageDescriptorSamplers</code> or <code>maxPerStageDescriptorUpdateAfterBindSamplers</code>	sampler
	combined image sampler
<code>maxPerStageDescriptorSampledImages</code> or <code>maxPerStageDescriptorUpdateAfterBindSampledImages</code>	sampled image
	combined image sampler
	uniform texel buffer
<code>maxPerStageDescriptorStorageImages</code> or <code>maxPerStageDescriptorUpdateAfterBindStorageImages</code>	storage image
	storage texel buffer
<code>maxPerStageDescriptorUniformBuffers</code> or <code>maxPerStageDescriptorUpdateAfterBindUniformBuffers</code>	uniform buffer
	uniform buffer dynamic
<code>maxPerStageDescriptorStorageBuffers</code> or <code>maxPerStageDescriptorUpdateAfterBindStorageBuffers</code>	storage buffer
	storage buffer dynamic
<code>maxPerStageDescriptorInputAttachments</code> or <code>maxPerStageDescriptorUpdateAfterBindInputAttachments</code>	input attachment ¹

1

Input attachments **can** only be used in the fragment shader stage

16.5.4. Offset and Stride Assignment

When a SPIR-V object is declared using an [explicit layout](#), it **must** be laid out according to the following additional requirements.

NOTE The numeric order of `Offset` decorations does not need to follow member declaration order.

Alignment Requirements

There are different alignment requirements depending on the specific resources and on the features enabled.

Matrix types are defined in terms of arrays as follows:

- A column-major matrix with C columns and R rows is equivalent to a C element array of vectors with R components.
- A row-major matrix with C columns and R rows is equivalent to an R element array of vectors with C components.

The *scalar alignment* of the type of an `OpTypeStruct` member is defined recursively as follows:

- A scalar of size N has a scalar alignment of N.
- A vector type has a scalar alignment equal to that of its component type.
- An array type has a scalar alignment equal to that of its element type.
- A structure has a scalar alignment equal to the largest scalar alignment of any of its members.
- A matrix type inherits *scalar alignment* from the equivalent array declaration.

The *base alignment* of the type of an `OpTypeStruct` member is defined recursively as follows:

- A scalar has a base alignment equal to its scalar alignment.
- A two-component vector has a base alignment equal to twice its scalar alignment.
- A three- or four-component vector has a base alignment equal to four times its scalar alignment.
- An array has a base alignment equal to the base alignment of its element type.
- A structure has a base alignment equal to the largest base alignment of any of its members. An empty structure has a base alignment equal to the size of the smallest scalar type permitted by the capabilities declared in the SPIR-V module. (e.g., for a 1 byte aligned empty structure in the `StorageBuffer` storage class, `StorageBuffer8BitAccess` or `UniformAndStorageBuffer8BitAccess` **must** be declared in the SPIR-V module.)
- A matrix type inherits *base alignment* from the equivalent array declaration.

The *extended alignment* of the type of an `OpTypeStruct` member is similarly defined as follows:

- A scalar or vector type has an extended alignment equal to its base alignment.
- An array or structure type has an extended alignment equal to the largest extended alignment of any of its members, rounded up to a multiple of 16.
- A matrix type inherits extended alignment from the equivalent array declaration.

A member is defined to *improperly straddle* if either of the following are true:

- It is a vector with total size less than or equal to 16 bytes, and has `Offset` decorations placing its first byte at F and its last byte at L, where $\text{floor}(F / 16) \neq \text{floor}(L / 16)$.
- It is a vector with total size greater than 16 bytes and has its `Offset` decorations placing its first byte at a non-integer multiple of 16.

Standard Buffer Layout

Every member of an `OpTypeStruct` that is required to be explicitly laid out **must** be aligned according to the first matching rule as follows. If the structure is contained in pointer types of multiple storage classes, it **must** satisfy the requirements for every storage class used to reference it.

1. If the `scalarBlockLayout` feature is enabled and the storage class is `Uniform`, `StorageBuffer`, `PhysicalStorageBuffer`, or `PushConstant` then every member **must** be aligned according to its scalar alignment.

2. All vectors **must** be aligned according to their scalar alignment.
3. If the `uniformBufferStandardLayout` feature is not enabled, then any member of an `OpTypeStruct` with a storage class of `Uniform` and a decoration of `Block` **must** be aligned according to its extended alignment.
4. Every other member **must** be aligned according to its base alignment.

NOTE

Even if scalar alignment is supported, it is generally more performant to use the *base alignment*.

The memory layout **must** obey the following rules:

- The `Offset` decoration of any member **must** be a multiple of its alignment.
- Any `ArrayStride` or `MatrixStride` decoration **must** be a multiple of the alignment of the array or matrix as defined above.

If one of the conditions below applies

- The storage class is `Uniform`, `StorageBuffer`, `PhysicalStorageBuffer`, or `PushConstant`, and the `scalarBlockLayout` feature is not enabled.
- The storage class is any other storage class.

the memory layout **must** also obey the following rules:

- Vectors **must** not improperly straddle, as defined above.
- The `Offset` decoration of a member **must** not place it between the end of a structure, an array or a matrix and the next multiple of the alignment of that structure, array or matrix.

NOTE

The **std430 layout** in GLSL satisfies these rules for types using the base alignment. The **std140 layout** satisfies the rules for types using the extended alignment.

16.6. Built-In Variables

Built-in variables are accessed in shaders by declaring a variable decorated with a `BuiltIn` SPIR-V decoration. The meaning of each `BuiltIn` decoration is as follows. In the remainder of this section, the name of a built-in is used interchangeably with a term equivalent to a variable decorated with that particular built-in. Built-ins that represent integer values **can** be declared as either signed or unsigned 32-bit integers.

[As mentioned above](#), some inputs and outputs have an additional level of arrayness relative to other shader inputs and outputs. This level of arrayness is not included in the type descriptions below, but **must** be included when declaring the built-in.

Any two variables declared in the `Input` storage class listed as operands on the same `OpEntryPoint` **must** not have the same `BuiltIn` decoration. Any two variables declared in the `Output` storage class listed as operands on the same `OpEntryPoint` **must** not have the same `BuiltIn` decoration.

BaseInstance

Decorating a variable with the **BaseInstance** built-in will make that variable contain the integer value corresponding to the first instance that was passed to the command that invoked the current vertex shader invocation. **BaseInstance** is the **firstInstance** parameter to a *direct drawing command* or the **firstInstance** member of a structure consumed by an *indirect drawing command*.

Valid Usage

- VUID-BaseInstance-BaseInstance-04181
The **BaseInstance** decoration **must** be used only within the **Vertex Execution Model**
- VUID-BaseInstance-BaseInstance-04182
The variable decorated with **BaseInstance** **must** be declared using the **Input Storage Class**
- VUID-BaseInstance-BaseInstance-04183
The variable decorated with **BaseInstance** **must** be declared as a scalar 32-bit integer value

BaseVertex

Decorating a variable with the **BaseVertex** built-in will make that variable contain the integer value corresponding to the first vertex or vertex offset that was passed to the command that invoked the current vertex shader invocation. For *non-indexed drawing commands*, this variable is the **firstVertex** parameter to a *direct drawing command* or the **firstVertex** member of the structure consumed by an *indirect drawing command*. For *indexed drawing commands*, this variable is the **vertexOffset** parameter to a *direct drawing command* or the **vertexOffset** member of the structure consumed by an *indirect drawing command*.

Valid Usage

- VUID-BaseVertex-BaseVertex-04184
The **BaseVertex** decoration **must** be used only within the **Vertex Execution Model**
- VUID-BaseVertex-BaseVertex-04185
The variable decorated with **BaseVertex** **must** be declared using the **Input Storage Class**
- VUID-BaseVertex-BaseVertex-04186
The variable decorated with **BaseVertex** **must** be declared as a scalar 32-bit integer value

ClipDistance

Decorating a variable with the **ClipDistance** built-in decoration will make that variable contain the mechanism for controlling user clipping. **ClipDistance** is an array such that the i^{th} element of the array specifies the clip distance for plane i . A clip distance of 0 means the vertex is on the plane, a positive distance means the vertex is inside the clip half-space, and a negative distance means the vertex is outside the clip half-space.

NOTE The array variable decorated with **ClipDistance** is explicitly sized by the shader.

NOTE

In the last [pre-rasterization shader stage](#), these values will be linearly interpolated across the primitive and the portion of the primitive with interpolated distances less than 0 will be considered outside the clip volume. If [ClipDistance](#) is then used by a fragment shader, [ClipDistance](#) contains these linearly interpolated values.

Valid Usage

- VUID-ClipDistance-ClipDistance-04187
The [ClipDistance](#) decoration **must** be used only within the [MeshEXT](#), [MeshNV](#), [Vertex](#), [Fragment](#), [TessellationControl](#), [TessellationEvaluation](#), or [Geometry Execution Model](#)
- VUID-ClipDistance-ClipDistance-04188
The variable decorated with [ClipDistance](#) within the [MeshEXT](#), [MeshNV](#), or [Vertex Execution Model](#) **must** be declared using the [Output Storage Class](#)
- VUID-ClipDistance-ClipDistance-04189
The variable decorated with [ClipDistance](#) within the [Fragment Execution Model](#) **must** be declared using the [Input Storage Class](#)
- VUID-ClipDistance-ClipDistance-04190
The variable decorated with [ClipDistance](#) within the [TessellationControl](#), [TessellationEvaluation](#), or [Geometry Execution Model](#) **must** not be declared in a [Storage Class](#) other than [Input](#) or [Output](#)
- VUID-ClipDistance-ClipDistance-04191
The variable decorated with [ClipDistance](#) **must** be declared as an array of 32-bit floating-point values

CullDistance

Decorating a variable with the [CullDistance](#) built-in decoration will make that variable contain the mechanism for controlling user culling. If any member of this array is assigned a negative value for all vertices belonging to a primitive, then the primitive is discarded before rasterization.

NOTE

In fragment shaders, the values of the [CullDistance](#) array are linearly interpolated across each primitive.

NOTE

If [CullDistance](#) decorates an input variable, that variable will contain the corresponding value from the [CullDistance](#) decorated output variable from the previous shader stage.

Valid Usage

- VUID-CullDistance-CullDistance-04196
The [CullDistance](#) decoration **must** be used only within the [MeshEXT](#), [MeshNV](#), [Vertex](#), [Fragment](#), [TessellationControl](#), [TessellationEvaluation](#), or [Geometry Execution Model](#)
- VUID-CullDistance-CullDistance-04197

The variable decorated with **CullDistance** within the **MeshEXT**, **MeshNV** or **Vertex Execution Model** **must** be declared using the **Output Storage Class**

- VUID-CullDistance-CullDistance-04198

The variable decorated with **CullDistance** within the **Fragment Execution Model** **must** be declared using the **Input Storage Class**

- VUID-CullDistance-CullDistance-04199

The variable decorated with **CullDistance** within the **TessellationControl**, **TessellationEvaluation**, or **Geometry Execution Model** **must** not be declared using a **Storage Class** other than **Input** or **Output**

- VUID-CullDistance-CullDistance-04200

The variable decorated with **CullDistance** **must** be declared as an array of 32-bit floating-point values

DeviceIndex

The **DeviceIndex** decoration **can** be applied to a shader input which will be filled with the device index of the physical device that is executing the current shader invocation. This value will be in the range $[0, \max(1, \text{physicalDeviceCount})]$, where **physicalDeviceCount** is the **physicalDeviceCount** member of **VkDeviceGroupDeviceCreateInfo**.

Valid Usage

- VUID-DeviceIndex-DeviceIndex-04205

The variable decorated with **DeviceIndex** **must** be declared using the **Input Storage Class**

- VUID-DeviceIndex-DeviceIndex-04206

The variable decorated with **DeviceIndex** **must** be declared as a scalar 32-bit integer value

DrawIndex

Decorating a variable with the **DrawIndex** built-in will make that variable contain the integer value corresponding to the zero-based index of the draw that invoked the current vertex shader invocation. For *indirect drawing commands*, **DrawIndex** begins at zero and increments by one for each draw executed. The number of draws is given by the **drawCount** parameter. For *direct drawing commands*, **DrawIndex** is always zero. **DrawIndex** is dynamically uniform.

Valid Usage

- VUID-DrawIndex-DrawIndex-04207

The **DrawIndex** decoration **must** be used only within the **Vertex**, **MeshEXT**, **TaskEXT**, **MeshNV**, or **TaskNV Execution Model**

- VUID-DrawIndex-DrawIndex-04208

The variable decorated with **DrawIndex** **must** be declared using the **Input Storage Class**

- VUID-DrawIndex-DrawIndex-04209

The variable decorated with **DrawIndex** **must** be declared as a scalar 32-bit integer value

FragCoord

Decorating a variable with the **FragCoord** built-in decoration will make that variable contain the coordinates (x,y,z,1/w) of the fragment being processed.

The (x,y) values are the framebuffer coordinates ($x_{f,y,f}$) of the fragment.

When **Sample Shading** is enabled, the x and y components of **FragCoord** reflect the location of one of the samples corresponding to the shader invocation.

Otherwise, the x and y components of **FragCoord** reflect the location of the center of the fragment.

The z component of **FragCoord** is the interpolated depth value of the primitive.

The w component is the interpolated $\frac{1}{w}$.

The **Centroid** interpolation decoration is ignored, but allowed, on **FragCoord**.

Valid Usage

- VUID-FragCoord-FragCoord-04210
The **FragCoord** decoration **must** be used only within the **Fragment Execution Model**
- VUID-FragCoord-FragCoord-04211
The variable decorated with **FragCoord** **must** be declared using the **Input Storage Class**
- VUID-FragCoord-FragCoord-04212
The variable decorated with **FragCoord** **must** be declared as a four-component vector of 32-bit floating-point values

FragDepth

To have a shader supply a fragment-depth value, the shader **must** declare the **DepthReplacing** execution mode. Such a shader's fragment-depth value will come from the variable decorated with the **FragDepth** built-in decoration.

This value will be used for any subsequent depth testing performed by the implementation or writes to the depth attachment. See [fragment shader depth replacement](#) for details.

Valid Usage

- VUID-FragDepth-FragDepth-04213
The **FragDepth** decoration **must** be used only within the **Fragment Execution Model**
- VUID-FragDepth-FragDepth-04214
The variable decorated with **FragDepth** **must** be declared using the **Output Storage Class**
- VUID-FragDepth-FragDepth-04215
The variable decorated with **FragDepth** **must** be declared as a scalar 32-bit floating-point value
- VUID-FragDepth-FragDepth-04216

If the shader dynamically writes to the variable decorated with **FragDepth**, the **DepthReplacing Execution Mode** **must** be declared

FragStencilRefEXT

Decorating a variable with the **FragStencilRefEXT** built-in decoration will make that variable contain the new stencil reference value for all samples covered by the fragment. This value will be used as the stencil reference value used in stencil testing.

To write to **FragStencilRefEXT**, a shader **must** declare the **StencilRefReplacingEXT** execution mode. If a shader declares the **StencilRefReplacingEXT** execution mode and there is an execution path through the shader that does not set **FragStencilRefEXT**, then the fragment's stencil reference value is undefined for executions of the shader that take that path.

Only the least significant **s** bits of the integer value of the variable decorated with **FragStencilRefEXT** are considered for stencil testing, where **s** is the number of bits in the stencil framebuffer attachment, and higher order bits are discarded.

See [fragment shader stencil reference replacement](#) for more details.

Valid Usage

- VUID-FragStencilRefEXT-FragStencilRefEXT-04223
The **FragStencilRefEXT** decoration **must** be used only within the **Fragment Execution Model**
- VUID-FragStencilRefEXT-FragStencilRefEXT-04224
The variable decorated with **FragStencilRefEXT** **must** be declared using the **Output Storage Class**
- VUID-FragStencilRefEXT-FragStencilRefEXT-04225
The variable decorated with **FragStencilRefEXT** **must** be declared as a scalar integer value

FrontFacing

Decorating a variable with the **FrontFacing** built-in decoration will make that variable contain whether the fragment is front or back facing. This variable is non-zero if the current fragment is considered to be part of a **front-facing** polygon primitive or of a non-polygon primitive and is zero if the fragment is considered to be part of a back-facing polygon primitive.

Valid Usage

- VUID-FrontFacing-FrontFacing-04229
The **FrontFacing** decoration **must** be used only within the **Fragment Execution Model**
- VUID-FrontFacing-FrontFacing-04230
The variable decorated with **FrontFacing** **must** be declared using the **Input Storage Class**
- VUID-FrontFacing-FrontFacing-04231
The variable decorated with **FrontFacing** **must** be declared as a boolean value

FullyCoveredEXT

Decorating a variable with the **FullyCoveredEXT** built-in decoration will make that variable indicate whether the **fragment area** is fully covered by the generating primitive. This variable is non-zero if conservative rasterization is enabled and the current fragment area is fully covered by the generating primitive, and is zero if the fragment is not covered or partially covered, or conservative rasterization is disabled.

Valid Usage

- VUID-FullyCoveredEXT-FullyCoveredEXT-04232
The **FullyCoveredEXT** decoration **must** be used only within the **Fragment Execution Model**
- VUID-FullyCoveredEXT-FullyCoveredEXT-04233
The variable decorated with **FullyCoveredEXT** **must** be declared using the **Input Storage Class**
- VUID-FullyCoveredEXT-FullyCoveredEXT-04234
The variable decorated with **FullyCoveredEXT** **must** be declared as a boolean value
- VUID-FullyCoveredEXT-conservativeRasterizationPostDepthCoverage-04235
If **VkPhysicalDeviceConservativeRasterizationPropertiesEXT::conservativeRasterizationPostDepthCoverage** is not supported the **PostDepthCoverage Execution Mode** **must** not be declared, when a variable with the **FullyCoveredEXT** decoration is declared

GlobalInvocationId

Decorating a variable with the **GlobalInvocationId** built-in decoration will make that variable contain the location of the current invocation within the global workgroup. Each component is equal to the index of the local workgroup multiplied by the size of the local workgroup plus **LocalInvocationId**.

Valid Usage

- VUID-GlobalInvocationId-GlobalInvocationId-04236
The **GlobalInvocationId** decoration **must** be used only within the **GLCompute, MeshEXT, TaskEXT, MeshNV, or TaskNV Execution Model**
- VUID-GlobalInvocationId-GlobalInvocationId-04237
The variable decorated with **GlobalInvocationId** **must** be declared using the **Input Storage Class**
- VUID-GlobalInvocationId-GlobalInvocationId-04238
The variable decorated with **GlobalInvocationId** **must** be declared as a three-component vector of 32-bit integer values

HelperInvocation

Decorating a variable with the **HelperInvocation** built-in decoration will make that variable contain whether the current invocation is a helper invocation. This variable is non-zero if the

current fragment being shaded is a helper invocation and zero otherwise. A helper invocation is an invocation of the shader that is produced to satisfy internal requirements such as the generation of derivatives.

NOTE

It is very likely that a helper invocation will have a value of `SampleMask` fragment shader input value that is zero.

Valid Usage

- VUID-HelperInvocation-HelperInvocation-04239
The `HelperInvocation` decoration **must** be used only within the `Fragment Execution Model`
- VUID-HelperInvocation-HelperInvocation-04240
The variable decorated with `HelperInvocation` **must** be declared using the `Input Storage Class`
- VUID-HelperInvocation-HelperInvocation-04241
The variable decorated with `HelperInvocation` **must** be declared as a boolean value

InvocationId

Decorating a variable with the `InvocationId` built-in decoration will make that variable contain the index of the current shader invocation in a geometry shader, or the index of the output patch vertex in a tessellation control shader.

In a geometry shader, the index of the current shader invocation ranges from zero to the number of `instances` declared in the shader minus one. If the instance count of the geometry shader is one or is not specified, then `InvocationId` will be zero.

Valid Usage

- VUID-InvocationId-InvocationId-04257
The `InvocationId` decoration **must** be used only within the `TessellationControl` or `Geometry Execution Model`
- VUID-InvocationId-InvocationId-04258
The variable decorated with `InvocationId` **must** be declared using the `Input Storage Class`
- VUID-InvocationId-InvocationId-04259
The variable decorated with `InvocationId` **must** be declared as a scalar 32-bit integer value

InstanceIndex

Decorating a variable in a vertex shader with the `InstanceIndex` built-in decoration will make that variable contain the index of the instance that is being processed by the current vertex shader invocation. `InstanceIndex` begins at the `firstInstance` parameter to `vkCmdDraw` or `vkCmdDrawIndexed` or at the `firstInstance` member of a structure consumed by `vkCmdDrawIndirect` or `vkCmdDrawIndexedIndirect`.

Valid Usage

- VUID-InstanceIndex-InstanceIndex-04263
The **InstanceIndex** decoration **must** be used only within the **Vertex Execution Model**
- VUID-InstanceIndex-InstanceIndex-04264
The variable decorated with **InstanceIndex** **must** be declared using the **Input Storage Class**
- VUID-InstanceIndex-InstanceIndex-04265
The variable decorated with **InstanceIndex** **must** be declared as a scalar 32-bit integer value

Layer

Decorating a variable with the **Layer** built-in decoration will make that variable contain the select layer of a multi-layer framebuffer attachment.

In a vertex, tessellation evaluation, or geometry shader, any variable decorated with **Layer** can be written with the framebuffer layer index to which the primitive produced by that shader will be directed.

The last active **pre-rasterization shader stage** (in pipeline order) controls the **Layer** that is used. Outputs in previous shader stages are not used, even if the last stage fails to write the **Layer**.

If the last active **pre-rasterization shader stage** shader entry point's interface does not include a variable decorated with **Layer**, then the first layer is used. If a **pre-rasterization shader stage** shader entry point's interface includes a variable decorated with **Layer**, it **must** write the same value to **Layer** for all output vertices of a given primitive. If the **Layer** value is less than 0 or greater than or equal to the number of layers in the framebuffer, then primitives **may** still be rasterized, fragment shaders **may** be executed, and the framebuffer values for all layers are undefined.

In a fragment shader, a variable decorated with **Layer** contains the layer index of the primitive that the fragment invocation belongs to.

Valid Usage

- VUID-Layer-Layer-04272
The **Layer** decoration **must** be used only within the **MeshEXT**, **MeshNV**, **Vertex**, **TessellationEvaluation**, **Geometry**, or **Fragment Execution Model**
- VUID-Layer-Layer-04273
If the **shaderOutputLayer** feature is not enabled then the **Layer** decoration **must** be used only within the **Geometry** or **Fragment Execution Model**
- VUID-Layer-Layer-04274
The variable decorated with **Layer** within the **MeshEXT**, **MeshNV**, **Vertex**, **TessellationEvaluation**, or **Geometry Execution Model** **must** be declared using the **Output Storage Class**
- VUID-Layer-Layer-04275

The variable decorated with **Layer** within the **Fragment Execution Model** **must** be declared using the **Input Storage Class**

- VUID-Layer-Layer-04276

The variable decorated with **Layer** **must** be declared as a scalar 32-bit integer value for all supported execution models except **MeshEXT**

- VUID-Layer-Layer-07039

The variable decorated with **Layer** within the **MeshEXT Execution Model** **must** also be decorated with the **PerPrimitiveEXT** decoration

- VUID-Layer-Layer-10592

Layer within the **MeshEXT Execution Model** **must** decorate a scalar 32-bit integer member of a structure decorated as **Block**, or decorate a variable of type **OpTypeArray** of scalar 32-bit integer values

- VUID-Layer-Layer-10593

If **Layer** is declared as an array of 32-bit integer values, within the **MeshEXT Execution Model**, size of the array **must** match the value specified by **OutputPrimitivesEXT**

- VUID-Layer-Layer-10594

If **Layer** decorates a member of a structure, the variable declaration of the containing **Block** type **must** have an array size that matches the value specified by **OutputPrimitivesEXT**

LocalInvocationId

Decorating a variable with the **LocalInvocationId** built-in decoration will make that variable contain the location of the current compute shader invocation within the local workgroup. Each component ranges from zero through to the size of the workgroup in that dimension minus one.

NOTE

If the size of the workgroup in a particular dimension is one, then the **LocalInvocationId** in that dimension will be zero. If the workgroup is effectively two-dimensional, then **LocalInvocationId.z** will be zero. If the workgroup is effectively one-dimensional, then both **LocalInvocationId.y** and **LocalInvocationId.z** will be zero.

Valid Usage

- VUID-LocalInvocationId-LocalInvocationId-04281

The **LocalInvocationId** decoration **must** be used only within the **GLCompute**, **MeshEXT**, **TaskEXT**, **MeshNV**, or **TaskNV Execution Model**

- VUID-LocalInvocationId-LocalInvocationId-04282

The variable decorated with **LocalInvocationId** **must** be declared using the **Input Storage Class**

- VUID-LocalInvocationId-LocalInvocationId-04283

The variable decorated with **LocalInvocationId** **must** be declared as a three-component vector of 32-bit integer values

LocalInvocationIndex

Decorating a variable with the **LocalInvocationIndex** built-in decoration will make that variable contain a one-dimensional representation of **LocalInvocationId**. This is computed as:

```
LocalInvocationIndex =  
    LocalInvocationId.z * WorkgroupSize.x * WorkgroupSize.y +  
    LocalInvocationId.y * WorkgroupSize.x +  
    LocalInvocationId.x;
```

Valid Usage

- VUID-LocalInvocationIndex-LocalInvocationIndex-04284
The **LocalInvocationIndex** decoration **must** be used only within the **GLCompute**, **MeshEXT**, **TaskEXT**, **MeshNV**, or **TaskNV Execution Model**
- VUID-LocalInvocationIndex-LocalInvocationIndex-04285
The variable decorated with **LocalInvocationIndex** **must** be declared using the **Input Storage Class**
- VUID-LocalInvocationIndex-LocalInvocationIndex-04286
The variable decorated with **LocalInvocationIndex** **must** be declared as a scalar 32-bit integer value

NumSubgroups

Decorating a variable with the **NumSubgroups** built-in decoration will make that variable contain the number of subgroups in the local workgroup.

Valid Usage

- VUID-NumSubgroups-NumSubgroups-04293
The **NumSubgroups** decoration **must** be used only within the **GLCompute**, **MeshEXT**, **TaskEXT**, **MeshNV**, or **TaskNV Execution Model**
- VUID-NumSubgroups-NumSubgroups-04294
The variable decorated with **NumSubgroups** **must** be declared using the **Input Storage Class**
- VUID-NumSubgroups-NumSubgroups-04295
The variable decorated with **NumSubgroups** **must** be declared as a scalar 32-bit integer value

NumWorkgroups

Decorating a variable with the **NumWorkgroups** built-in decoration will make that variable contain the number of local workgroups that are part of the dispatch that the invocation belongs to. Each component is equal to the values of the workgroup count parameters passed into the dispatching commands.

Valid Usage

- VUID-NumWorkgroups-NumWorkgroups-04296
The **NumWorkgroups** decoration **must** be used only within the **GLCompute**, **MeshEXT**, or **TaskEXT Execution Model**
- VUID-NumWorkgroups-NumWorkgroups-04297
The variable decorated with **NumWorkgroups** **must** be declared using the **Input Storage Class**
- VUID-NumWorkgroups-NumWorkgroups-04298
The variable decorated with **NumWorkgroups** **must** be declared as a three-component vector of 32-bit integer values

PatchVertices

Decorating a variable with the **PatchVertices** built-in decoration will make that variable contain the number of vertices in the input patch being processed by the shader. In a Tessellation Control Shader, this is the same as the `name:patchControlPoints` member of **VkPipelineTessellationStateCreateInfo**. In a Tessellation Evaluation Shader, **PatchVertices** is equal to the tessellation control output patch size. When the same shader is used in different pipelines where the patch sizes are configured differently, the value of the **PatchVertices** variable will also differ.

Valid Usage

- VUID-PatchVertices-PatchVertices-04308
The **PatchVertices** decoration **must** be used only within the **TessellationControl** or **TessellationEvaluation Execution Model**
- VUID-PatchVertices-PatchVertices-04309
The variable decorated with **PatchVertices** **must** be declared using the **Input Storage Class**
- VUID-PatchVertices-PatchVertices-04310
The variable decorated with **PatchVertices** **must** be declared as a scalar 32-bit integer value

PointCoord

Decorating a variable with the **PointCoord** built-in decoration will make that variable contain the coordinate of the current fragment within the point being rasterized, normalized to the size of the point with origin in the upper left corner of the point, as described in **Basic Point Rasterization**. If the primitive the fragment shader invocation belongs to is not a point, then the variable decorated with **PointCoord** contains an undefined value.

NOTE Depending on how the point is rasterized, **PointCoord** **may** never reach (0,0) or (1,1).

Valid Usage

- VUID-PointCoord-PointCoord-04311

The **PointCoord** decoration **must** be used only within the **Fragment Execution Model**

- VUID-PointCoord-PointCoord-04312

The variable decorated with **PointCoord** **must** be declared using the **Input Storage Class**

- VUID-PointCoord-PointCoord-04313

The variable decorated with **PointCoord** **must** be declared as a two-component vector of 32-bit floating-point values

PointSize

Decorating a variable with the **PointSize** built-in decoration will make that variable contain the size of point primitives. The value written to the variable decorated with **PointSize** by the last **pre-rasterization shader stage** in the pipeline is used as the framebuffer-space size of points produced by rasterization.

NOTE

When **PointSize** decorates a variable in the **Input Storage Class**, it contains the data written to the output variable decorated with **PointSize** from the previous shader stage.

Valid Usage

- VUID-PointSize-PointSize-04314

The **PointSize** decoration **must** be used only within the **MeshEXT**, **MeshNV**, **Vertex**, **TessellationControl**, **TessellationEvaluation**, or **Geometry Execution Model**

- VUID-PointSize-PointSize-04315

The variable decorated with **PointSize** within the **MeshEXT**, **MeshNV**, or **Vertex Execution Model** **must** be declared using the **Output Storage Class**

- VUID-PointSize-PointSize-04316

The variable decorated with **PointSize** within the **TessellationControl**, **TessellationEvaluation**, or **Geometry Execution Model** **must** not be declared using a **Storage Class** other than **Input** or **Output**

- VUID-PointSize-PointSize-04317

The variable decorated with **PointSize** **must** be declared as a scalar 32-bit floating-point value

Position

Decorating a variable with the **Position** built-in decoration will make that variable contain the position of the current vertex. In the last **pre-rasterization shader stage**, the value of the variable decorated with **Position** is used in subsequent primitive assembly, clipping, and rasterization operations.

NOTE

When **Position** decorates a variable in the **Input Storage Class**, it contains the data written to the output variable decorated with **Position** from the previous shader stage.

Valid Usage

- VUID-Position-Position-04318
The **Position** decoration **must** be used only within the **MeshEXT**, **MeshNV**, **Vertex**, **TessellationControl**, **TessellationEvaluation**, or **Geometry Execution Model**
- VUID-Position-Position-04319
The variable decorated with **Position** within the **MeshEXT**, **MeshNV**, or **Vertex Execution Model** **must** be declared using the **Output Storage Class**
- VUID-Position-Position-04320
The variable decorated with **Position** within the **TessellationControl**, **TessellationEvaluation**, or **Geometry Execution Model** **must** not be declared using a **Storage Class** other than **Input** or **Output**
- VUID-Position-Position-04321
The variable decorated with **Position** **must** be declared as a four-component vector of 32-bit floating-point values

PrimitiveId

Decorating a variable with the **PrimitiveId** built-in decoration will make that variable contain the index of the current primitive.

The index of the first primitive generated by a drawing command is zero, and the index is incremented after every individual point, line, or triangle primitive is processed.

For triangles drawn as points or line segments (see **Polygon Mode**), the primitive index is incremented only once, even if multiple points or lines are eventually drawn.

Variables decorated with **PrimitiveId** are reset to zero between each instance drawn.

Restarting a primitive topology using primitive restart has no effect on the value of variables decorated with **PrimitiveId**.

In tessellation control and tessellation evaluation shaders, it will contain the index of the patch within the current set of rendering primitives that corresponds to the shader invocation.

In a geometry shader, it will contain the number of primitives presented as input to the shader since the current set of rendering primitives was started.

In a fragment shader, it will contain the primitive index written by the geometry shader if a geometry shader is present, or with the value that would have been presented as input to the geometry shader had it been present.

NOTE

When the **PrimitiveId** decoration is applied to an output variable in the geometry shader, the resulting value is seen through the **PrimitiveId** decorated input variable in the fragment shader.

The fragment shader using **PrimitiveId** will need to declare either the **Geometry** or **Tessellation** capability to satisfy the requirement SPIR-V has to use **PrimitiveId**.

Valid Usage

- VUID-PrimitiveId-PrimitiveId-04330
The **PrimitiveId** decoration **must** be used only within the **MeshEXT**, **MeshNV**, **IntersectionKHR**, **AnyHitKHR**, **ClosestHitKHR**, **TessellationControl**, **TessellationEvaluation**, **Geometry**, or **Fragment Execution Model**
- VUID-PrimitiveId-Fragment-04331
If pipeline contains both the **Fragment** and **Geometry Execution Model** and a variable decorated with **PrimitiveId** is read from **Fragment** shader, then the **Geometry** shader **must** write to the output variables decorated with **PrimitiveId** in all execution paths
- VUID-PrimitiveId-Fragment-04332
If pipeline contains both the **Fragment** and **MeshEXT** or **MeshNV Execution Model** and a variable decorated with **PrimitiveId** is read from **Fragment** shader, then the **MeshEXT** or **MeshNV** shader **must** write to the output variables decorated with **PrimitiveId** in all execution paths
- VUID-PrimitiveId-Fragment-04333
If **Fragment Execution Model** contains a variable decorated with **PrimitiveId**, then either the **MeshShadingEXT**, **MeshShadingNV**, **Geometry** or **Tessellation** capability **must** also be declared
- VUID-PrimitiveId-PrimitiveId-04334
The variable decorated with **PrimitiveId** within the **TessellationControl**, **TessellationEvaluation**, **Fragment**, **IntersectionKHR**, **AnyHitKHR**, or **ClosestHitKHR Execution Model** **must** be declared using the **Input Storage Class**
- VUID-PrimitiveId-PrimitiveId-04335
The variable decorated with **PrimitiveId** within the **Geometry Execution Model** **must** be declared using the **Input** or **Output Storage Class**
- VUID-PrimitiveId-PrimitiveId-04336
The variable decorated with **PrimitiveId** within the **MeshEXT** or **MeshNV Execution Model** **must** be declared using the **Output Storage Class**
- VUID-PrimitiveId-PrimitiveId-04337
The variable decorated with **PrimitiveId** **must** be declared as a scalar 32-bit integer value for all supported execution models except **MeshEXT**
- VUID-PrimitiveId-PrimitiveId-07040
The variable decorated with **PrimitiveId** within the **MeshEXT Execution Model** **must** also be decorated with the **PerPrimitiveEXT** decoration
- VUID-PrimitiveId-PrimitiveId-10595
PrimitiveId within the **MeshEXT Execution Model** **must** decorate a scalar 32-bit integer member of a structure decorated as **Block**, or decorate a variable of type **OpTypeArray** of 32-bit integer values
- VUID-PrimitiveId-PrimitiveId-10596
If **PrimitiveId** is declared as an array of 32-bit integer values, within the **MeshEXT Execution Model**, size of the array **must** match the value specified by **OutputPrimitivesEXT**
- VUID-PrimitiveId-PrimitiveId-10597
If **PrimitiveId** decorates a member of a structure, the variable declaration of the

containing **Block** type **must** have an array size that matches the value specified by **OutputPrimitivesEXT**

PrimitiveShadingRateKHR

Decorating a variable with the **PrimitiveShadingRateKHR** built-in decoration will make that variable contain the **primitive fragment shading rate**.

The value written to the variable decorated with **PrimitiveShadingRateKHR** by the last **pre-rasterization shader stage** in the pipeline is used as the **primitive fragment shading rate**. Outputs in previous shader stages are ignored.

If the last active **pre-rasterization shader stage** shader entry point's interface does not include a variable decorated with **PrimitiveShadingRateKHR**, then it is as if the shader specified a fragment shading rate value of 0, indicating a horizontal and vertical rate of 1 pixel.

If a shader has **PrimitiveShadingRateKHR** in the output interface and there is an execution path through the shader that does not write to it, its value is undefined for executions of the shader that take that path.

Valid Usage

- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-04484
The **PrimitiveShadingRateKHR** decoration **must** be used only within the **MeshEXT**, **MeshNV**, **Vertex**, or **Geometry Execution Model**
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-04485
The variable decorated with **PrimitiveShadingRateKHR** **must** be declared using the **Output Storage Class**
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-04486
The variable decorated with **PrimitiveShadingRateKHR** **must** be declared as a scalar 32-bit integer value for all supported execution models except **MeshEXT**
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-04487
The value written to **PrimitiveShadingRateKHR** **must** include no more than one of **Vertical2Pixels** and **Vertical4Pixels**
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-04488
The value written to **PrimitiveShadingRateKHR** **must** include no more than one of **Horizontal2Pixels** and **Horizontal4Pixels**
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-04489
The value written to **PrimitiveShadingRateKHR** **must** not have any bits set other than those defined by **Fragment Shading Rate Flags** enumerants in the SPIR-V specification
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-07059
The variable decorated with **PrimitiveShadingRateKHR** within the **MeshEXT Execution Model** **must** also be decorated with the **PerPrimitiveEXT** decoration
- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-10598
PrimitiveShadingRateKHR within the **MeshEXT Execution Model** **must** decorate a scalar 32-bit integer member of a structure decorated as **Block**, or decorate a variable of type

OpTypeArray of 32-bit integer values

- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-10599

If **PrimitiveShadingRateKHR** is declared as an array of 32-bit integer values, within the **MeshEXT Execution Model**, size of the array **must** match the value specified by **OutputPrimitivesEXT**

- VUID-PrimitiveShadingRateKHR-PrimitiveShadingRateKHR-10600

If **PrimitiveShadingRateKHR** decorates a member of a structure, the variable declaration of the containing **Block** type **must** have an array size that matches the value specified by **OutputPrimitivesEXT**

SampleId

Decorating a variable with the **SampleId** built-in decoration will make that variable contain the **coverage index** for the current fragment shader invocation. **SampleId** ranges from zero to the number of samples in the framebuffer minus one. If a fragment shader entry point's interface includes an input variable decorated with **SampleId**, **Sample Shading** is considered enabled with a **minSampleShading** value of 1.0.

Valid Usage

- VUID-SampleId-SampleId-04354

The **SampleId** decoration **must** be used only within the **Fragment Execution Model**

- VUID-SampleId-SampleId-04355

The variable decorated with **SampleId** **must** be declared using the **Input Storage Class**

- VUID-SampleId-SampleId-04356

The variable decorated with **SampleId** **must** be declared as a scalar 32-bit integer value

SampleMask

Decorating a variable with the **SampleMask** built-in decoration will make any variable contain the **sample mask** for the current fragment shader invocation.

A variable in the **Input** storage class decorated with **SampleMask** will contain a bitmask of the set of samples covered by the primitive generating the fragment during rasterization. It has a sample bit set if and only if the sample is considered covered for this fragment shader invocation. **SampleMask[]** is an array of integers. Bits are mapped to samples in a manner where bit B of mask M (**SampleMask[M]**) corresponds to sample $32 \times M + B$.

A variable in the **Output** storage class decorated with **SampleMask** is an array of integers forming a bit array in a manner similar to an input variable decorated with **SampleMask**, but where each bit represents coverage as computed by the shader. This computed **SampleMask** is combined with the generated coverage mask in the **multisample coverage** operation.

Variables decorated with **SampleMask** **must** be either an unsized array, or explicitly sized to be no larger than the implementation-dependent maximum sample-mask (as an array of 32-bit elements), determined by the maximum number of samples.

If a fragment shader entry point's interface includes an output variable decorated with **SampleMask**, the sample mask will be undefined for any array elements of any fragment shader invocations that fail to assign a value. If a fragment shader entry point's interface does not include an output variable decorated with **SampleMask**, the sample mask has no effect on the processing of a fragment.

Valid Usage

- VUID-SampleMask-SampleMask-04357
The **SampleMask** decoration **must** be used only within the **Fragment Execution Model**
- VUID-SampleMask-SampleMask-04358
The variable decorated with **SampleMask** **must** be declared using the **Input** or **Output Storage Class**
- VUID-SampleMask-SampleMask-04359
The variable decorated with **SampleMask** **must** be declared as an array of 32-bit integer values

SamplePosition

Decorating a variable with the **SamplePosition** built-in decoration will make that variable contain the sub-pixel position of the sample being shaded. The top left of the pixel is considered to be at coordinate (0,0) and the bottom right of the pixel is considered to be at coordinate (1,1).

If a fragment shader entry point's interface includes an input variable decorated with **SamplePosition**, **Sample Shading** is considered enabled with a **minSampleShading** value of 1.0.

If the current pipeline uses **custom sample locations** the value of any variable decorated with the **SamplePosition** built-in decoration is undefined.

Valid Usage

- VUID-SamplePosition-SamplePosition-04360
The **SamplePosition** decoration **must** be used only within the **Fragment Execution Model**
- VUID-SamplePosition-SamplePosition-04361
The variable decorated with **SamplePosition** **must** be declared using the **Input Storage Class**
- VUID-SamplePosition-SamplePosition-04362
The variable decorated with **SamplePosition** **must** be declared as a two-component vector of 32-bit floating-point values

ShadingRateKHR

Decorating a variable with the **ShadingRateKHR** built-in decoration will make that variable contain the **fragment shading rate** for the current fragment invocation.

Valid Usage

- VUID-ShadingRateKHR-ShadingRateKHR-04490
The **ShadingRateKHR** decoration **must** be used only within the **Fragment Execution Model**
- VUID-ShadingRateKHR-ShadingRateKHR-04491
The variable decorated with **ShadingRateKHR** **must** be declared using the **Input Storage Class**
- VUID-ShadingRateKHR-ShadingRateKHR-04492
The variable decorated with **ShadingRateKHR** **must** be declared as a scalar 32-bit integer value

SubgroupId

Decorating a variable with the **SubgroupId** built-in decoration will make that variable contain the index of the subgroup within the local workgroup. This variable is in range [0, **NumSubgroups**-1].

Valid Usage

- VUID-SubgroupId-SubgroupId-04367
The **SubgroupId** decoration **must** be used only within the **GLCompute**, **MeshEXT**, **TaskEXT**, **MeshNV**, or **TaskNV Execution Model**
- VUID-SubgroupId-SubgroupId-04368
The variable decorated with **SubgroupId** **must** be declared using the **Input Storage Class**
- VUID-SubgroupId-SubgroupId-04369
The variable decorated with **SubgroupId** **must** be declared as a scalar 32-bit integer value

SubgroupEqMask

Decorating a variable with the **SubgroupEqMask** builtin decoration will make that variable contain the *subgroup mask* of the current subgroup invocation. The bit corresponding to the **SubgroupLocalInvocationId** is set in the variable decorated with **SubgroupEqMask**. All other bits are set to zero.

SubgroupEqMaskKHR is an alias of **SubgroupEqMask**.

Valid Usage

- VUID-SubgroupEqMask-SubgroupEqMask-04370
The variable decorated with **SubgroupEqMask** **must** be declared using the **Input Storage Class**
- VUID-SubgroupEqMask-SubgroupEqMask-04371
The variable decorated with **SubgroupEqMask** **must** be declared as a four-component vector of 32-bit integer values

SubgroupGeMask

Decorating a variable with the `SubgroupGeMask` builtin decoration will make that variable contain the *subgroup mask* of the current subgroup invocation. The bits corresponding to the invocations greater than or equal to `SubgroupLocalInvocationId` through `SubgroupSize-1` are set in the variable decorated with `SubgroupGeMask`. All other bits are set to zero.

`SubgroupGeMaskKHR` is an alias of `SubgroupGeMask`.

Valid Usage

- VUID-SubgroupGeMask-SubgroupGeMask-04372
The variable decorated with `SubgroupGeMask` **must** be declared using the `Input Storage Class`
- VUID-SubgroupGeMask-SubgroupGeMask-04373
The variable decorated with `SubgroupGeMask` **must** be declared as a four-component vector of 32-bit integer values

SubgroupGtMask

Decorating a variable with the `SubgroupGtMask` builtin decoration will make that variable contain the *subgroup mask* of the current subgroup invocation. The bits corresponding to the invocations greater than `SubgroupLocalInvocationId` through `SubgroupSize-1` are set in the variable decorated with `SubgroupGtMask`. All other bits are set to zero.

`SubgroupGtMaskKHR` is an alias of `SubgroupGtMask`.

Valid Usage

- VUID-SubgroupGtMask-SubgroupGtMask-04374
The variable decorated with `SubgroupGtMask` **must** be declared using the `Input Storage Class`
- VUID-SubgroupGtMask-SubgroupGtMask-04375
The variable decorated with `SubgroupGtMask` **must** be declared as a four-component vector of 32-bit integer values

SubgroupLeMask

Decorating a variable with the `SubgroupLeMask` builtin decoration will make that variable contain the *subgroup mask* of the current subgroup invocation. The bits corresponding to the invocations less than or equal to `SubgroupLocalInvocationId` are set in the variable decorated with `SubgroupLeMask`. All other bits are set to zero.

`SubgroupLeMaskKHR` is an alias of `SubgroupLeMask`.

Valid Usage

- VUID-SubgroupLeMask-SubgroupLeMask-04376

The variable decorated with **SubgroupLeMask** **must** be declared using the **Input Storage Class**

- VUID-SubgroupLeMask-SubgroupLeMask-04377

The variable decorated with **SubgroupLeMask** **must** be declared as a four-component vector of 32-bit integer values

SubgroupLtMask

Decorating a variable with the **SubgroupLtMask** builtin decoration will make that variable contain the *subgroup mask* of the current subgroup invocation. The bits corresponding to the invocations less than **SubgroupLocalInvocationId** are set in the variable decorated with **SubgroupLtMask**. All other bits are set to zero.

SubgroupLtMaskKHR is an alias of **SubgroupLtMask**.

Valid Usage

- VUID-SubgroupLtMask-SubgroupLtMask-04378

The variable decorated with **SubgroupLtMask** **must** be declared using the **Input Storage Class**

- VUID-SubgroupLtMask-SubgroupLtMask-04379

The variable decorated with **SubgroupLtMask** **must** be declared as a four-component vector of 32-bit integer values

SubgroupLocalInvocationId

Decorating a variable with the **SubgroupLocalInvocationId** builtin decoration will make that variable contain the index of the invocation within the subgroup. This variable is in range $[0, \text{SubgroupSize}-1]$.

If **VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT** is specified, full subgroups are enabled for that pipeline stage. When full subgroups are enabled, subgroups **must** be launched with all invocations active, i.e., there is an active invocation with **SubgroupLocalInvocationId** for each value in range $[0, \text{SubgroupSize}-1]$.

NOTE

There is no direct relationship between **SubgroupLocalInvocationId** and **LocalInvocationId** or **LocalInvocationIndex**. If the pipeline was created with full subgroups applications can compute their own local invocation index to serve the same purpose:

$$\text{index} = \text{SubgroupLocalInvocationId} + \text{SubgroupId} \times \text{SubgroupSize}$$

If full subgroups are not enabled, some subgroups may be dispatched with inactive invocations that do not correspond to a local workgroup invocation, making the value of index unreliable.

Valid Usage

- VUID-SubgroupLocalInvocationId-SubgroupLocalInvocationId-04380
The variable decorated with **SubgroupLocalInvocationId** **must** be declared using the **Input Storage Class**
- VUID-SubgroupLocalInvocationId-SubgroupLocalInvocationId-04381
The variable decorated with **SubgroupLocalInvocationId** **must** be declared as a scalar 32-bit integer value

SubgroupSize

Decorating a variable with the **SubgroupSize** builtin decoration will make that variable contain the implementation-dependent **number of invocations in a subgroup**. This value **must** be a power-of-two integer.

If the pipeline was created with the **VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT** flag set, the **SubgroupSize** decorated variable will contain the subgroup size for each subgroup that gets dispatched. This value **must** be between **minSubgroupSize** and **maxSubgroupSize** and **must** be uniform with **subgroup scope**. The value **may** vary across a single draw call, and for fragment shaders **may** vary across a single primitive. In compute dispatches, **SubgroupSize** **must** be uniform with **command scope**.

If the pipeline was created with a chained **VkPipelineShaderStageRequiredSubgroupSizeCreateInfo** structure, the **SubgroupSize** decorated variable will match **requiredSubgroupSize**.

If the pipeline was not created with the **VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT** flag set and no **VkPipelineShaderStageRequiredSubgroupSizeCreateInfo** structure was chained, the variable decorated with **SubgroupSize** will match **subgroupSize**.

The maximum number of invocations that an implementation can support per subgroup is 128.

Valid Usage

- VUID-SubgroupSize-SubgroupSize-04382
The variable decorated with **SubgroupSize** **must** be declared using the **Input Storage Class**
- VUID-SubgroupSize-SubgroupSize-04383
The variable decorated with **SubgroupSize** **must** be declared as a scalar 32-bit integer value

TessCoord

Decorating a variable with the **TessCoord** built-in decoration will make that variable contain the three-dimensional (u,v,w) barycentric coordinate of the tessellated vertex within the patch. u, v, and w are in the range [0,1] and vary linearly across the primitive being subdivided. For the

tessellation modes of **Quads** or **Isolines**, the third component is always zero.

Valid Usage

- VUID-TessCoord-TessCoord-04387
The **TessCoord** decoration **must** be used only within the **TessellationEvaluation Execution Model**
- VUID-TessCoord-TessCoord-04388
The variable decorated with **TessCoord** **must** be declared using the **Input Storage Class**
- VUID-TessCoord-TessCoord-04389
The variable decorated with **TessCoord** **must** be declared as a three-component vector of 32-bit floating-point values

TessLevelOuter

Decorating a variable with the **TessLevelOuter** built-in decoration will make that variable contain the outer tessellation levels for the current patch.

In tessellation control shaders, the variable decorated with **TessLevelOuter** **can** be written to, controlling the tessellation factors for the resulting patch. These values are used by the tessellator to control primitive tessellation and **can** be read by tessellation evaluation shaders.

In tessellation evaluation shaders, the variable decorated with **TessLevelOuter** **can** read the values written by the tessellation control shader.

Valid Usage

- VUID-TessLevelOuter-TessLevelOuter-04390
The **TessLevelOuter** decoration **must** be used only within the **TessellationControl** or **TessellationEvaluation Execution Model**
- VUID-TessLevelOuter-TessLevelOuter-04391
The variable decorated with **TessLevelOuter** within the **TessellationControl Execution Model** **must** be declared using the **Output Storage Class**
- VUID-TessLevelOuter-TessLevelOuter-04392
The variable decorated with **TessLevelOuter** within the **TessellationEvaluation Execution Model** **must** be declared using the **Input Storage Class**
- VUID-TessLevelOuter-TessLevelOuter-04393
The variable decorated with **TessLevelOuter** **must** be declared as an array of size four, containing 32-bit floating-point values

TessLevelInner

Decorating a variable with the **TessLevelInner** built-in decoration will make that variable contain the inner tessellation levels for the current patch.

In tessellation control shaders, the variable decorated with **TessLevelInner** **can** be written to,

controlling the tessellation factors for the resulting patch. These values are used by the tessellator to control primitive tessellation and **can** be read by tessellation evaluation shaders.

In tessellation evaluation shaders, the variable decorated with **TessLevelInner** **can** read the values written by the tessellation control shader.

Valid Usage

- VUID-TessLevelInner-TessLevelInner-04394
The **TessLevelInner** decoration **must** be used only within the **TessellationControl** or **TessellationEvaluation** Execution Model
- VUID-TessLevelInner-TessLevelInner-04395
The variable decorated with **TessLevelInner** within the **TessellationControl** Execution Model **must** be declared using the **Output Storage Class**
- VUID-TessLevelInner-TessLevelInner-04396
The variable decorated with **TessLevelInner** within the **TessellationEvaluation** Execution Model **must** be declared using the **Input Storage Class**
- VUID-TessLevelInner-TessLevelInner-04397
The variable decorated with **TessLevelInner** **must** be declared as an array of size two, containing 32-bit floating-point values

VertexIndex

Decorating a variable with the **VertexIndex** built-in decoration will make that variable contain the index of the vertex that is being processed by the current vertex shader invocation. For non-indexed draws, this variable begins at the **firstVertex** parameter to **vkCmdDraw** or the **firstVertex** member of a structure consumed by **vkCmdDrawIndirect** and increments by one for each vertex in the draw. For indexed draws, its value is the content of the index buffer for the vertex plus the **vertexOffset** parameter to **vkCmdDrawIndexed** or the **vertexOffset** member of the structure consumed by **vkCmdDrawIndexedIndirect**.

NOTE **VertexIndex** starts at the same starting value for each instance.

Valid Usage

- VUID-VertexIndex-VertexIndex-04398
The **VertexIndex** decoration **must** be used only within the **Vertex** Execution Model
- VUID-VertexIndex-VertexIndex-04399
The variable decorated with **VertexIndex** **must** be declared using the **Input Storage Class**
- VUID-VertexIndex-VertexIndex-04400
The variable decorated with **VertexIndex** **must** be declared as a scalar 32-bit integer value

ViewIndex

The **ViewIndex** decoration **can** be applied to a shader input which will be filled with the index of the view that is being processed by the current shader invocation.

If multiview is enabled in the render pass, this value will be the index of one of the bits set in the view mask of the subpass the pipeline is compiled against. If multiview is not enabled in the render pass, this value will be zero.

Valid Usage

- VUID-ViewIndex-ViewIndex-04401
The **ViewIndex** decoration **must** be used only within the **MeshEXT**, **Vertex**, **Geometry**, **TessellationControl**, **TessellationEvaluation** or **Fragment Execution Model**
- VUID-ViewIndex-ViewIndex-04402
The variable decorated with **ViewIndex** **must** be declared using the **Input Storage Class**
- VUID-ViewIndex-ViewIndex-04403
The variable decorated with **ViewIndex** **must** be declared as a scalar 32-bit integer value

ViewportIndex

Decorating a variable with the **ViewportIndex** built-in decoration will make that variable contain the index of the viewport.

In a vertex, tessellation evaluation, or geometry shader, the variable decorated with **ViewportIndex** can be written to with the viewport index to which the primitive produced by that shader will be directed.

The selected viewport index is used to select the viewport transform and scissor rectangle.

The last active *pre-rasterization shader stage* (in pipeline order) controls the **ViewportIndex** that is used. Outputs in previous shader stages are not used, even if the last stage fails to write the **ViewportIndex**.

If the last active *pre-rasterization shader stage* shader entry point's interface does not include a variable decorated with **ViewportIndex** then the first viewport is used. If a *pre-rasterization shader stage* shader entry point's interface includes a variable decorated with **ViewportIndex**, it **must** write the same value to **ViewportIndex** for all output vertices of a given primitive.

In a fragment shader, the variable decorated with **ViewportIndex** contains the viewport index of the primitive that the fragment invocation belongs to.

Valid Usage

- VUID-ViewportIndex-ViewportIndex-04404
The **ViewportIndex** decoration **must** be used only within the **MeshEXT**, **MeshNV**, **Vertex**, **TessellationEvaluation**, **Geometry**, or **Fragment Execution Model**
- VUID-ViewportIndex-ViewportIndex-04405
If the **shaderOutputViewportIndex** feature is not enabled then the **ViewportIndex** decoration **must** be used only within the **Geometry** or **Fragment Execution Model**
- VUID-ViewportIndex-ViewportIndex-04406
The variable decorated with **ViewportIndex** within the **MeshEXT**, **MeshNV**, **Vertex**,

TessellationEvaluation, or **Geometry Execution Model** **must** be declared using the **Output Storage Class**

- VUID-ViewportIndex-ViewportIndex-04407
The variable decorated with **ViewportIndex** within the **Fragment Execution Model** **must** be declared using the **Input Storage Class**
- VUID-ViewportIndex-ViewportIndex-04408
The variable decorated with **ViewportIndex** **must** be declared as a scalar 32-bit integer value for all supported execution models except **MeshEXT**
- VUID-ViewportIndex-ViewportIndex-07060
The variable decorated with **ViewportIndex** within the **MeshEXT Execution Model** **must** also be decorated with the **PerPrimitiveEXT** decoration
- VUID-ViewportIndex-ViewportIndex-10601
ViewportIndex within the **MeshEXT Execution Model** **must** decorate a scalar 32-bit integer member of a structure decorated as **Block**, or decorate a variable of type **OpTypeArray** of scalar 32-bit integer values
- VUID-ViewportIndex-ViewportIndex-10602
If **ViewportIndex** is declared as an array of 32-bit integer values, within the **MeshEXT Execution Model**, size of the array **must** match the value specified by **OutputPrimitivesEXT**
- VUID-ViewportIndex-ViewportIndex-10603
If **ViewportIndex** decorates a member of a structure, the variable declaration of the containing **Block** type **must** have an array size that matches the value specified by **OutputPrimitivesEXT**

WorkgroupId

Decorating a variable with the **WorkgroupId** built-in decoration will make that variable contain the global coordinate of the local workgroup that the current invocation is a member of. Each component is in the range [base,base + count), where base and count are based on the parameters passed into the dispatching commands in each dimension.

Valid Usage

- VUID-WorkgroupId-WorkgroupId-04422
The **WorkgroupId** decoration **must** be used only within the **GLCompute**, **MeshEXT**, **TaskEXT**, **MeshNV**, or **TaskNV Execution Model**
- VUID-WorkgroupId-WorkgroupId-04423
The variable decorated with **WorkgroupId** **must** be declared using the **Input Storage Class**
- VUID-WorkgroupId-WorkgroupId-04424
The variable decorated with **WorkgroupId** **must** be declared as a three-component vector of 32-bit integer values

WorkgroupSize

Decorating an object with the **WorkgroupSize** built-in decoration will make that object contain the dimensions of a local workgroup. If an object is decorated with the **WorkgroupSize** decoration, this

takes precedence over any **LocalSize** execution mode.

Valid Usage

- VUID-WorkgroupSize-WorkgroupSize-04425
The **WorkgroupSize** decoration **must** be used only within the **GLCompute**, **MeshEXT**, **TaskEXT**, **MeshNV**, or **TaskNV Execution Model**
- VUID-WorkgroupSize-WorkgroupSize-04426
The variable decorated with **WorkgroupSize** **must** be a specialization constant or a constant
- VUID-WorkgroupSize-WorkgroupSize-04427
The variable decorated with **WorkgroupSize** **must** be declared as a three-component vector of 32-bit integer values

Chapter 17. Image Operations

17.1. Image Operations Overview

Vulkan Image Operations are operations performed on **images** by SPIR-V Image Instructions which take an **OpTypeImage** (representing a **VkImageView**) or **OpTypeSampledImage** (representing a (**VkImageView**, **VkSampler**) pair). Read, write, and atomic operations also take texel coordinates as operands, and return a value based on a neighborhood of texture elements (*texels*) within the image. Query operations return properties of the bound image or of the lookup itself. The “Depth” operand of **OpTypeImage** is ignored.

Image Operations include the functionality of the following SPIR-V Image Instructions:

- **OpImageSample*** and **OpImageSparseSample*** read one or more neighboring texels of the image, and **filter** the texel values based on the state of the sampler.
 - Instructions with **ImplicitLod** in the name **determine** the LOD used in the sampling operation based on the coordinates used in neighboring fragments.
 - Instructions with **ExplicitLod** in the name **determine** the LOD used in the sampling operation based on additional coordinates.
 - Instructions with **Proj** in the name apply homogeneous **projection** to the coordinates.
- **OpImageFetch** and **OpImageSparseFetch** return a single texel of the image. No sampler is used.
- **OpImage*Gather** and **OpImageSparse*Gather** read neighboring texels and **return a single component** of each.
- **OpImageRead** (and **OpImageSparseRead**) and **OpImageWrite** read and write, respectively, a texel in the image. No sampler is used.
- **OpImage*Dref*** instructions apply **depth comparison** on the texel values.
- **OpImageSparse*** instructions additionally return a **sparse residency** code.
- **OpImageQuerySize**, **OpImageQuerySizeLod**, **OpImageQueryLevels**, and **OpImageQuerySamples** return properties of the image descriptor that would be accessed. The image itself is not accessed.
- **OpImageQueryLod** returns the LOD parameters that would be used in a sample operation. The actual operation is not performed.

17.1.1. Sampling Coordinate Systems

There are three *sampling coordinate systems* used in this chapter:

- normalized sampling coordinates [0.0, 1.0]
- unnormalized sampling coordinates [0.0, width / height / depth)
- integer sampling coordinates [0, width / height / depth)

SPIR-V **OpImageFetch**, **OpImageSparseFetch**, **OpImageRead**, **OpImageSparseRead**, and **OpImageWrite** instructions use integer sampling coordinates.

Other image instructions **can** use either normalized or unnormalized sampling coordinates (selected by the `unnormalizedCoordinates` state of the sampler used in the instruction), but there are **limitations** on what operations, image state, and sampler state is supported. Normalized coordinates are logically **converted** to unnormalized as part of image operations, and **certain steps** are only performed on normalized coordinates. The array layer coordinate is always treated as unnormalized even when other coordinates are normalized.

Normalized texel coordinates are referred to as (s,t,r,q,a), with the coordinates having the following meanings:

- s: Coordinate in the first dimension of an image.
- t: Coordinate in the second dimension of an image.
- r: Coordinate in the third dimension of an image.
 - (s,t,r) are interpreted as a direction vector for Cube images.
- q: Fourth coordinate, for homogeneous (projective) coordinates.
- a: Coordinate for array layer.

The coordinates are extracted from the SPIR-V operand based on the dimensionality of the image variable and type of instruction. For **Proj** instructions, the components are in order (s, [t,] [r,] q), with t and r being conditionally present based on the **Dim** of the image. For non-**Proj** instructions, the coordinates are (s [t] [r] [a]), with t and r being conditionally present based on the **Dim** of the image and a being conditionally present based on the **Arrayed** property of the image. Projective image instructions are not supported on **Arrayed** images.

Unnormalized texel coordinates are referred to as (u,v,w,a), with the coordinates having the following meanings:

- u: Coordinate in the first dimension of an image.
- v: Coordinate in the second dimension of an image.
- w: Coordinate in the third dimension of an image.
- a: Coordinate for array layer.

Only the u and v coordinates are directly extracted from the SPIR-V operand, because only 1D and 2D (non-**Arrayed**) dimensionalities support unnormalized coordinates. The components are in order (u [v]), with v being conditionally present when the dimensionality is 2D. When normalized coordinates are converted to unnormalized coordinates, all four coordinates are used.

Integer sampling coordinates are referred to as (i,j,k,l,n), with the coordinates having the following meanings:

- i: Coordinate in the first dimension of an image.
- j: Coordinate in the second dimension of an image.
- k: Coordinate in the third dimension of an image.
- l: Coordinate for array layer.
- n: Index of the sample within the texel.

They are extracted from the SPIR-V operand in order (i [j] [k] [l] [n]), with j and k conditionally present based on the **Dim** of the image, and l conditionally present based on the **Arrayed** property of the image. n is conditionally present and is taken from the **Sample** image operand.

Final integer coordinates are used as **image coordinates** to perform an **image read** after sampling calculations, directly translating each coordinate as follows:

- $i \rightarrow x$
- $j \rightarrow y$
- $k \rightarrow z$
- $l \rightarrow \text{layer}$
- $n \rightarrow \text{sample}$

level is calculated separately via the **Lod** image operand if present, or is set to 0 otherwise.

For all coordinate types, unused coordinates are assigned a value of zero.

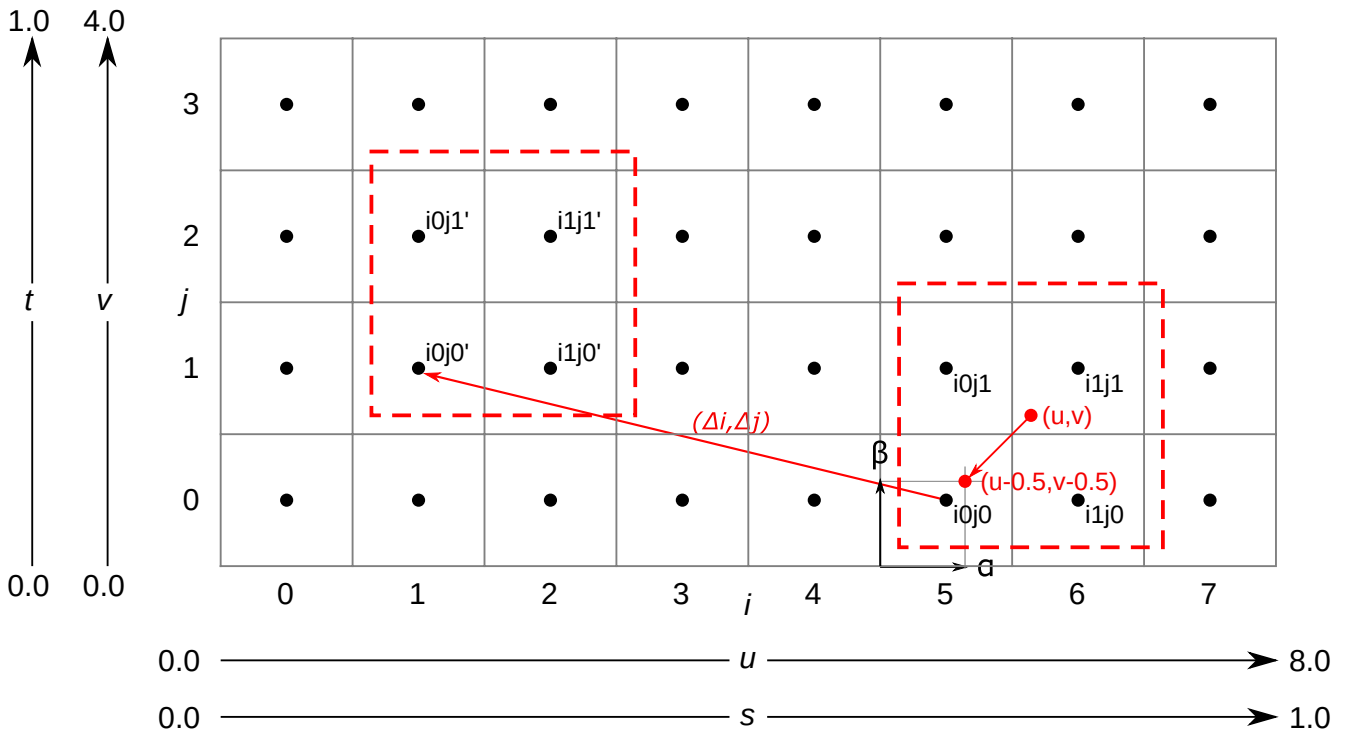


Figure 3. Texel Coordinate Systems, Linear Filtering

The Texel Coordinate Systems - For the example shown of an 8×4 texel two dimensional image.

- Normalized texel coordinates:
 - The s coordinate goes from 0.0 to 1.0.
 - The t coordinate goes from 0.0 to 1.0.
- Unnormalized texel coordinates:
 - The u coordinate within the range 0.0 to 8.0 is within the image, otherwise it is outside the image.
 - The v coordinate within the range 0.0 to 4.0 is within the image, otherwise it is outside the

image.

- Integer texel coordinates:
 - The i coordinate within the range 0 to 7 addresses texels within the image, otherwise it is outside the image.
 - The j coordinate within the range 0 to 3 addresses texels within the image, otherwise it is outside the image.
- Also shown for linear filtering:
 - Given the unnormalized coordinates (u,v) , the four texels selected are i_0j_0 , i_1j_0 , i_0j_1 , and i_1j_1 .
 - The fractions α and β .
 - Given the offset Δ_i and Δ_j , the four texels selected by the offset are $i_0j'_0$, $i_1j'_0$, $i_0j'_1$, and $i_1j'_1$.

NOTE

For formats with reduced-resolution components, Δ_i and Δ_j are relative to the resolution of the highest-resolution component, and therefore may be divided by two relative to the unnormalized coordinate space of the lower-resolution components.

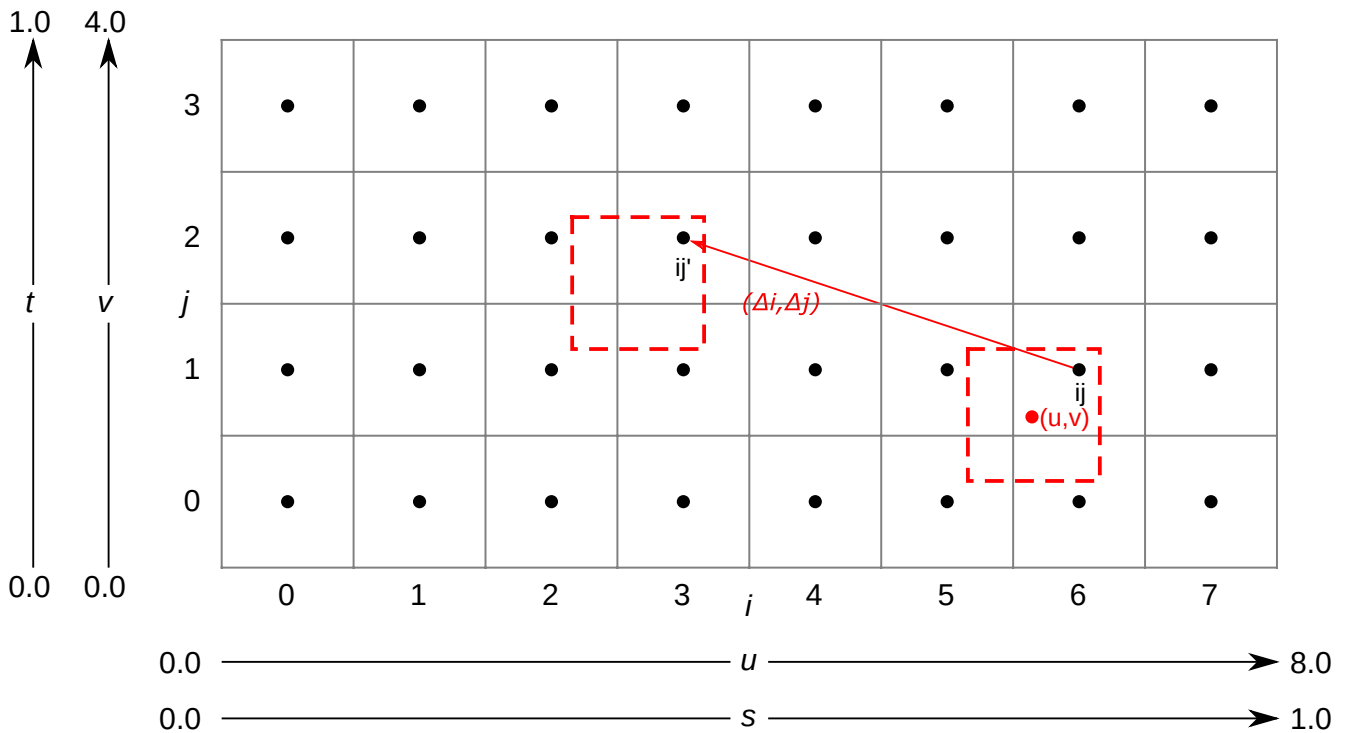


Figure 4. Texel Coordinate Systems, Nearest Filtering

The Texel Coordinate Systems - For the example shown of an 8x4 texel two dimensional image.

- Texel coordinates as above. Also shown for nearest filtering:
 - Given the unnormalized coordinates (u,v) , the texel selected is ij .
 - Given the offset Δ_i and Δ_j , the texel selected by the offset is ij' .

17.2. Sampling Operations

Sampling instructions are SPIR-V image instructions that read from an image with a sampler.

Sampling operations are a set of steps that are performed on state, coordinates, and texel values while processing a sampling instruction, and which are common to some or all sampling instructions. They include the following steps, which are performed in the listed order:

- Validation operations
 - Instruction/Sampler/Image validation
 - Coordinate validation
 - Layout validation
 - Cube Map Edge Handling
- Border Replacement
- Texel Reads
- Depth comparison
- Component swizzle
- Chroma reconstruction
- Y'C_BC_R conversion

For texel input instructions involving multiple texels (for sampling or gathering), these steps are applied for each texel that is used in the instruction. Depending on the type of image instruction, other steps are conditionally performed between these steps or involving multiple coordinate or texel values.

If [Chroma Reconstruction](#) is implicit, [Texel Filtering](#) instead takes place during chroma reconstruction, before [sampler Y'C_BC_R conversion](#) occurs.

17.2.1. Texel Input Validation Operations

Texel input validation operations inspect instruction/image/sampler state or coordinates, and in certain circumstances cause the texel value to be replaced or become undefined. There are a series of validations that the texel undergoes.

Instruction/Sampler/Image View Validation

There are a number of cases where a SPIR-V instruction **can** mismatch with the sampler, the image view, or both, and a number of further cases where the sampler **can** mismatch with the image view. In such cases the value of the texel returned is undefined.

These cases include:

- The sampler `borderColor` is an integer type and the image view `format` is not one of the [VkFormat](#) integer types or a stencil component of a depth/stencil format.
- The sampler `borderColor` is a float type and the image view `format` is not one of the [VkFormat](#) float types or a depth component of a depth/stencil format.
- The sampler `borderColor` is one of the opaque black colors (`VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK` or `VK_BORDER_COLOR_INT_OPAQUE_BLACK`) and the image view [VkComponentSwizzle](#) for any of the [VkComponentMapping](#) components is not the [identity swizzle](#).

- The sampler `borderColor` is a custom color (`VK_BORDER_COLOR_FLOAT_CUSTOM_EXT` or `VK_BORDER_COLOR_INT_CUSTOM_EXT`) and the supplied `VkSamplerCustomBorderColorCreateInfoEXT::customBorderColor` is outside the bounds of the values representable in the image view's `format`.
- The sampler `borderColor` is a custom color (`VK_BORDER_COLOR_FLOAT_CUSTOM_EXT` or `VK_BORDER_COLOR_INT_CUSTOM_EXT`) and the image view `VkComponentSwizzle` for any of the `VkComponentMapping` components is not the `identity swizzle`.
- The `VkImageLayout` of any subresource in the image view does not match the `VkDescriptorImageInfo::imageLayout` used to write the image descriptor.
- The SPIR-V Image Format is not `compatible` with the image view's `format`.
- The sampler `unnormalizedCoordinates` is `VK_TRUE` and any of the `limitations of unnormalized coordinates` are violated.
- The SPIR-V instruction is one of the `OpImage*Dref*` instructions and the sampler `compareEnable` is `VK_FALSE`
- The SPIR-V instruction is not one of the `OpImage*Dref*` instructions and the sampler `compareEnable` is `VK_TRUE`
- The SPIR-V instruction is one of the `OpImage*Dref*` instructions and the image view `format` is not one of the depth/stencil formats with a depth component, or the image view aspect is not `VK_IMAGE_ASPECT_DEPTH_BIT`.
- The SPIR-V instruction's image variable's properties are not compatible with the image view:
 - If the image view's `viewType` is one of `VK_IMAGE_VIEW_TYPE_1D_ARRAY`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY`, or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY` then the instruction **must** have `Arrayed` = 1. Otherwise the instruction **must** have `Arrayed` = 0.
 - If the image was created with `VkImageCreateInfo::samples` equal to `VK_SAMPLE_COUNT_1_BIT`, the instruction **must** have `MS` = 0.
 - If the image was created with `VkImageCreateInfo::samples` not equal to `VK_SAMPLE_COUNT_1_BIT`, the instruction **must** have `MS` = 1.
 - If the `Sampled Type` of the `OpTypeImage` does not match the `SPIR-V Type`.
 - If the `signedness of any read or sample operation` does not match the signedness of the image's format.
- The sampler was created with a specified `VkSamplerCustomBorderColorCreateInfoEXT::format` which does not match the `VkFormat` of the image view(s) it is sampling.
- The sampler is sampling an image view of `VK_FORMAT_B4G4R4A4_UNORM_PACK16`, `VK_FORMAT_B5G6R5_UNORM_PACK16`, or `VK_FORMAT_B5G5R5A1_UNORM_PACK16` format without a specified `VkSamplerCustomBorderColorCreateInfoEXT::format`.

Only `OpImageSample*` and `OpImageSparseSample*` **can** be used with a sampler or image view that enables `sampler Y'CBCR conversion`.

`OpImageFetch`, `OpImageSparseFetch`, `OpImage*Gather`, and `OpImageSparse*Gather` **must** not be used with a sampler or image view that enables `sampler Y'CBCR conversion`.

The `ConstOffset` and `Offset` operands **must** not be used with a sampler or image view that enables

sampler Y'C_BC_R conversion.

If the underlying **VkImage** format has an X component in its format description, undefined values are read from those bits.

NOTE

If the **VkImage** format and **VkImageView** format are the same, these bits will be unused by format conversion and this will have no effect. However, if the **VkImageView** format is different, then some bits of the result may be undefined. For example, when a **VK_FORMAT_R10X6_UNORM_PACK16** **VkImage** is sampled via a **VK_FORMAT_R16_UNORM** **VkImageView**, the low 6 bits of the value before format conversion are undefined and format conversion may return a range of different values.

NOTE

Some implementations will return undefined values in the case where a sampler uses a **VkSamplerAddressMode** of **VK_SAMPLER_ADDRESS_MODE_MIRRORED_REPEAT**, the sampler is used with operands **Offset**, **ConstOffset**, or **ConstOffsets**, and the value of the offset is larger than or equal to the corresponding width, height, or depth of any accessed image level.

This behavior was not tested prior to Vulkan conformance test suite version 1.3.8.0. Affected implementations will have a conformance test waiver for this issue.

Layout Validation

If all planes of a *disjoint multi-planar* image are not in the same **image layout**, the image **must** not be sampled with **sampler Y'C_BC_R conversion** enabled.

Coordinate Validation

Once the **normalized** or **unnormalized coordinates** have been converted to integer image coordinates, the integer coordinates are validated as image coordinates, as outlined in **Image Coordinate Validation**, converted as follows:

- $x = i$
- $y = j$
- $z = k$
- $\text{layer} = l$
- $\text{sample} = n$
- $\text{level} = d$

Cube Map Edge Handling

When sampling a cube map, if the image coordinates are out of bounds of the **selected cube map face**, the following steps are performed.

NOTE

This does not occur when using **VK_FILTER_NEAREST** filtering within a mip level, since **VK_FILTER_NEAREST** is treated as using **VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE**.

- Cube Map Edge Texel
 - If the texel lies beyond the selected cube map face in either only x or only y, then the coordinates (x,y,layer) are transformed to select the adjacent texel from the appropriate neighboring face.
- Cube Map Corner Texel
 - If the texel lies beyond the selected cube map face in both x and y, then there is no unique neighboring face from which to read that texel. The texel **should** be replaced by the average of the three values of the adjacent texels in each incident face. However, implementations **may** replace the cube map corner texel by other methods. The methods are subject to the constraint that for linear filtering if the three available texels have the same value, the resulting filtered texel **must** have that value, and for cubic filtering if the twelve available samples have the same value, the resulting filtered texel **must** have that value.

17.2.2. Border Replacement

If the sampler includes a border, out of bounds texels are replaced with a value based on the image format and the **borderColor** of the sampler. The border color is:

Table 20. Border Color B, Custom Border Color *VkSamplerCustomBorderColorCreateInfoEXT*
::customBorderColor U

Sampler borderColor	Corresponding Border Color
VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK	$[B_r, B_g, B_b, B_a] = [0.0, 0.0, 0.0, 0.0]$
VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK	$[B_r, B_g, B_b, B_a] = [0.0, 0.0, 0.0, 1.0]$
VK_BORDER_COLOR_FLOAT_OPAQUE_WHITE	$[B_r, B_g, B_b, B_a] = [1.0, 1.0, 1.0, 1.0]$
VK_BORDER_COLOR_INT_TRANSPARENT_BLACK	$[B_r, B_g, B_b, B_a] = [0, 0, 0, 0]$
VK_BORDER_COLOR_INT_OPAQUE_BLACK	$[B_r, B_g, B_b, B_a] = [0, 0, 0, 1]$
VK_BORDER_COLOR_INT_OPAQUE_WHITE	$[B_r, B_g, B_b, B_a] = [1, 1, 1, 1]$
VK_BORDER_COLOR_FLOAT_CUSTOM_EXT	$[B_r, B_g, B_b, B_a] = [U_r, U_g, U_b, U_a]$
VK_BORDER_COLOR_INT_CUSTOM_EXT	$[B_r, B_g, B_b, B_a] = [U_r, U_g, U_b, U_a]$

The custom border color (U) **may** be rounded by implementations prior to texel replacement, but the error introduced by such a rounding **must** not exceed one ULP of the image's **format**.

NOTE

The names **VK_BORDER_COLOR*_TRANSPARENT_BLACK**, **VK_BORDER_COLOR*_OPAQUE_BLACK**, and **VK_BORDER_COLOR*_OPAQUE_WHITE** are meant to describe which components are zeros and ones in the vocabulary of compositing, and are not meant to imply that the numerical value of **VK_BORDER_COLOR_INT_OPAQUE_WHITE** is a saturating value for integers.

This is substituted for the texel value by replacing the number of components in the image format

Table 21. Border Texel Components After Replacement

Texel Aspect or Format	Component Assignment
Depth aspect	$D = B_r$
Stencil aspect	$S = B_r^\dagger$
One component color format	$\text{Color}_r = B_r$
Two component color format	$[\text{Color}_r, \text{Color}_g] = [B_r, B_g]$
Three component color format	$[\text{Color}_r, \text{Color}_g, \text{Color}_b] = [B_r, B_g, B_b]$
Four component color format	$[\text{Color}_r, \text{Color}_g, \text{Color}_b, \text{Color}_a] = [B_r, B_g, B_b, B_a]$

$\dagger S = B_g$ **may** be substituted as the replacement method by the implementation when `VkSamplerCreateInfo::borderColor` is `VK_BORDER_COLOR_INT_CUSTOM_EXT` and `VkSamplerCustomBorderColorCreateInfoEXT::format` is `VK_FORMAT_UNDEFINED`. Implementations **should** use $S = B_r$ as the replacement method.

Implementations **may** swap the blue and alpha channels when sampling non-custom border colors with the `VK_FORMAT_B4G4R4A4_UNORM_PACK16` format, or the red and alpha channels with the `VK_FORMAT_R4G4B4A4_UNORM_PACK16` format.

NOTE

As `VK_FORMAT_B4G4R4A4_UNORM_PACK16` is required by Vulkan, support must be advertised for this format. Some Vulkan implementations on Apple hardware implement these formats through a hardware format with a different channel order, swizzled to match Vulkan's expectations. Unfortunately the swizzle cannot be readily applied to the fixed border colors - resulting in the apparent channel swap. For most standard border colors this does not result in a modification to the sampled output. However, `VK_BORDER_COLOR_*_OPAQUE_BLACK` will instead be sampled as transparent red or blue. If the `customBorderColorWithoutFormat` feature is supported and enabled, this functionality is expected to work without issue, but this feature may come with a performance cost.

When border color replacement occurs, [texel reads](#) are skipped, and the replaced color is used for ongoing operations instead.

17.2.3. Texel Reads

A texel is read from an image, performed as outlined in [Image Reads](#), using the converted image coordinates.

The returned components of each texel are then processed by further input operations.

17.2.4. Depth Compare Operation

If the image view has a depth/stencil format, the depth component is selected by the `aspectMask`, and the operation is an `OpImage*Dref*` instruction, a depth comparison is performed. The result is 1.0 if the comparison evaluates to true, and 0.0 otherwise. This value replaces the depth component D.

The compare operation is selected by the `VkCompareOp` value set by `VkSamplerCreateInfo::compareOp`. The reference value from the SPIR-V operand D_{ref} and the texel depth value D_{tex} are used

as the *reference* and *test* values, respectively, in that operation.

If the image being sampled has an unsigned normalized fixed-point format, then D_{ref} is clamped to [0,1] before the compare operation.

If the value of `magFilter` is `VK_FILTER_LINEAR`, or the value of `minFilter` is `VK_FILTER_LINEAR`, then D may be computed in an implementation-dependent manner which differs from the normal rules of linear filtering. The resulting value **must** be in the range [0,1] and should be proportional to, or a weighted average of, the number of comparison passes or failures.

17.2.5. Component Swizzle

All texel input instructions apply a *swizzle* based on:

- the `VkComponentSwizzle` enums in the `components` member of the `VkImageViewCreateInfo` structure for the image being read if `sampler Y'CBCR conversion` is not enabled, and
- the `VkComponentSwizzle` enums in the `components` member of the `VkSamplerYcbcrConversionCreateInfo` structure for the `sampler Y'CBCR conversion` if `sampler Y'CBCR conversion` is enabled.

The swizzle **can** rearrange the components of the texel, or substitute zero or one for any components. It is defined as follows for each color component:

$$Color'_{component} = \begin{cases} Color_r & \text{for RED swizzle} \\ Color_g & \text{for GREEN swizzle} \\ Color_b & \text{for BLUE swizzle} \\ Color_a & \text{for ALPHA swizzle} \\ 0 & \text{for ZERO swizzle} \\ one & \text{for ONE swizzle} \\ identity & \text{for IDENTITY swizzle} \end{cases}$$

where:

$$one = \begin{cases} 1.0f & \text{for floating-point components} \\ 1 & \text{for integer components} \end{cases}$$

$$identity = \begin{cases} Color_r & \text{for } component = r \\ Color_g & \text{for } component = g \\ Color_b & \text{for } component = b \\ Color_a & \text{for } component = a \end{cases}$$

If the border color is one of the `VK_BORDER_COLOR_*_OPAQUE_BLACK` enums and the `VkComponentSwizzle` is not the `identity swizzle` for all components, the value of the texel after swizzle is undefined.

If the image view has a depth/stencil format and the `VkComponentSwizzle` is `VK_COMPONENT_SWIZZLE_ONE`, the value of the texel after swizzle is undefined.

17.2.6. Sparse Residency

`OpImageSparse*` instructions return a structure which includes a *residency code* indicating whether any texels accessed by the instruction are sparse unbound texels. This code **can** be interpreted by the `OpImageSparseTexelsResident` instruction which converts the residency code to a boolean value.

17.2.7. Chroma Reconstruction

In some color models, the color representation is defined in terms of monochromatic light intensity (often called “luma”) and color differences relative to this intensity, often called “chroma”. It is common for color models other than RGB to represent the chroma components at lower spatial resolution than the luma component. This approach is used to take advantage of the eye’s lower spatial sensitivity to color compared with its sensitivity to brightness. Less commonly, the same approach is used with additive color, since the green component dominates the eye’s sensitivity to light intensity and the spatial sensitivity to color introduced by red and blue is lower.

Lower-resolution components are “downsampled” by resizing them to a lower spatial resolution than the component representing luminance. This process is also commonly known as “chroma subsampling”. There is one luminance sample in each texture texel, but each chrominance sample may be shared among several texels in one or both texture dimensions.

- “_444” formats do not spatially downsample chroma values compared with luma: there are unique chroma samples for each texel.
- “_422” formats have downsampling in the x dimension (corresponding to *u* or *s* coordinates): they are sampled at half the resolution of luma in that dimension.
- “_420” formats have downsampling in the x dimension (corresponding to *u* or *s* coordinates) and the y dimension (corresponding to *v* or *t* coordinates): they are sampled at half the resolution of luma in both dimensions.

The process of reconstructing a full color value for texture access involves accessing both chroma and luma values at the same location. To generate the color accurately, the values of the lower-resolution components at the location of the luma samples are reconstructed from the lower-resolution sample locations, an operation known here as “chroma reconstruction” irrespective of the actual color model.

The location of the chroma samples relative to the luma coordinates is determined by the `xChromaOffset` and `yChromaOffset` members of the `VkSamplerYcbcrConversionCreateInfo` structure used to create the sampler Y’C_BC_R conversion.

The following diagrams show the relationship between unnormalized (*u,v*) coordinates and (*i,j*) integer texel positions in the luma component (shown in black, with circles showing integer sample positions) and the texel coordinates of reduced-resolution chroma components, shown as crosses in red.

NOTE

If the chroma values are reconstructed at the locations of the luma samples by means of interpolation, chroma samples from outside the image bounds are needed; these are determined according to [Wrapping Operation](#). These diagrams represent this by showing the bounds of the “chroma texel” extending beyond the

image bounds, and including additional chroma sample positions where required for interpolation. The limits of a sample for **NEAREST** sampling is shown as a grid.

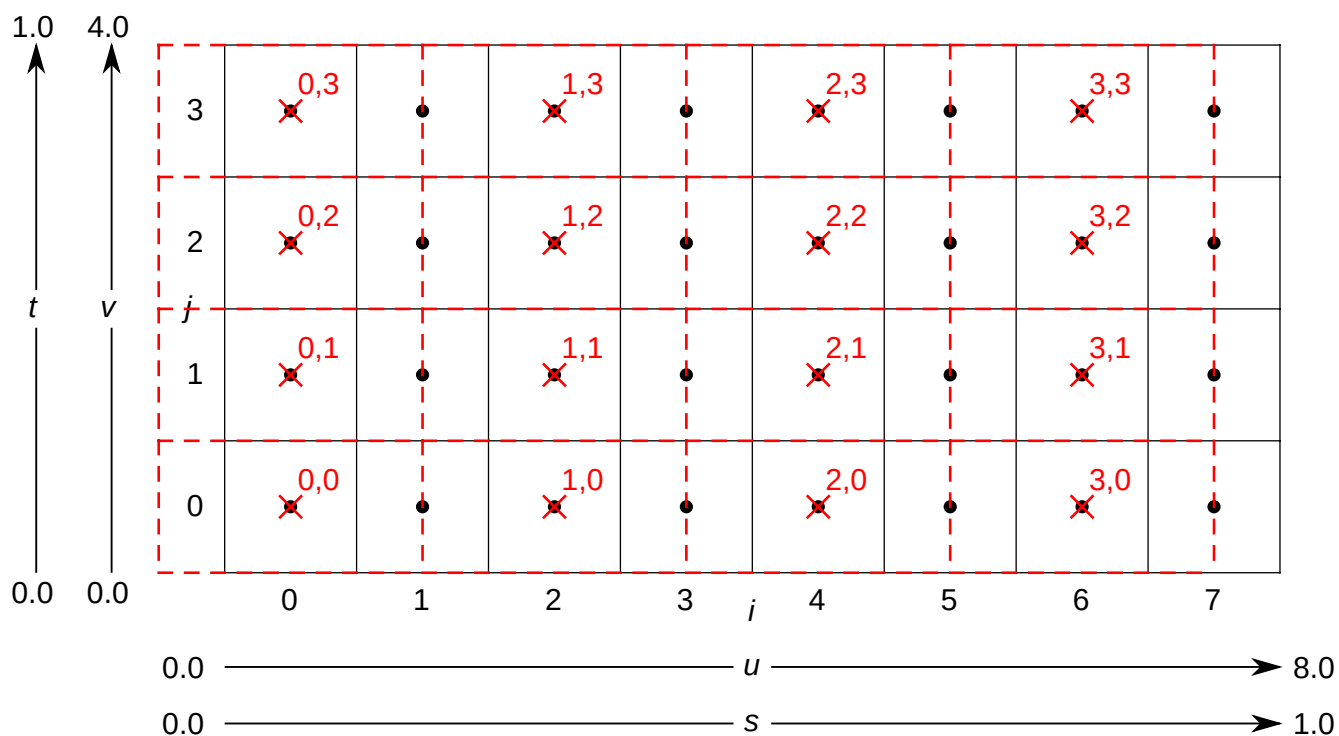


Figure 5. 422 downsampling, $xChromaOffset=COSITED\_EVEN$

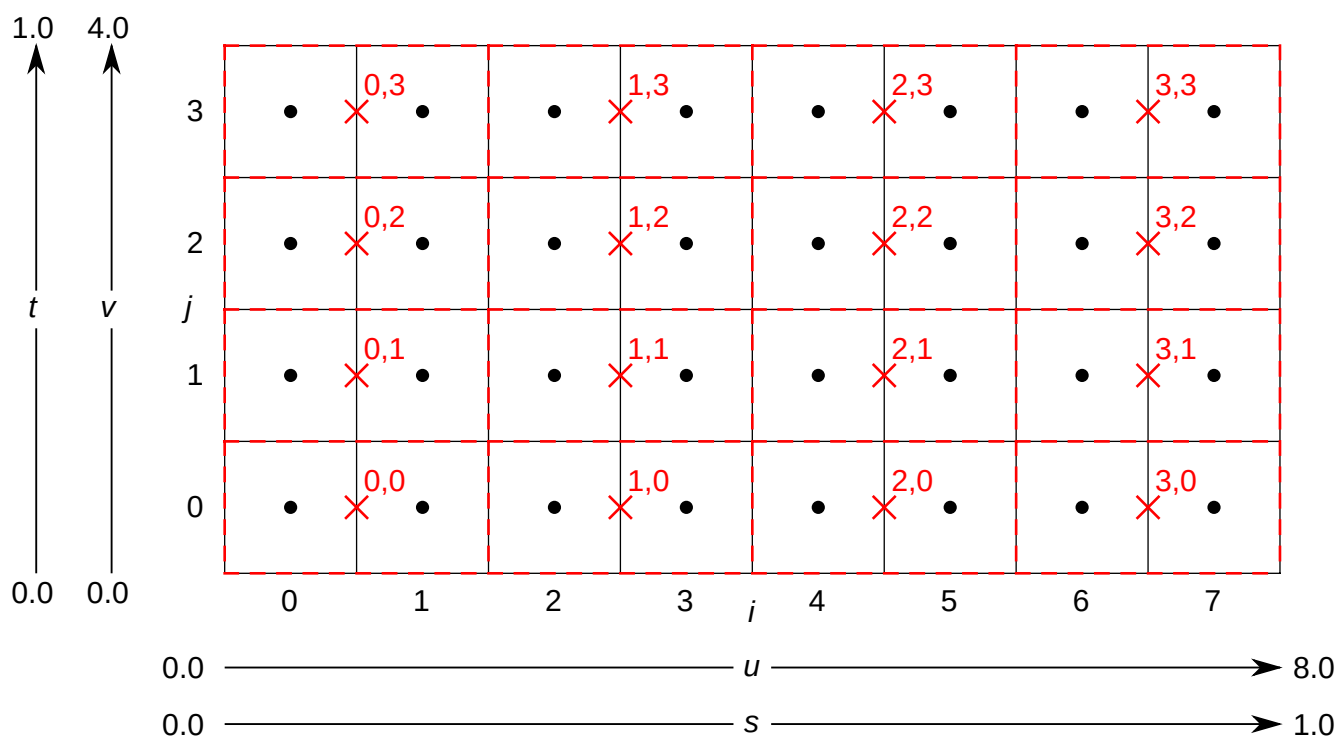


Figure 6. 422 downsampling, $xChromaOffset=MIDPOINT$

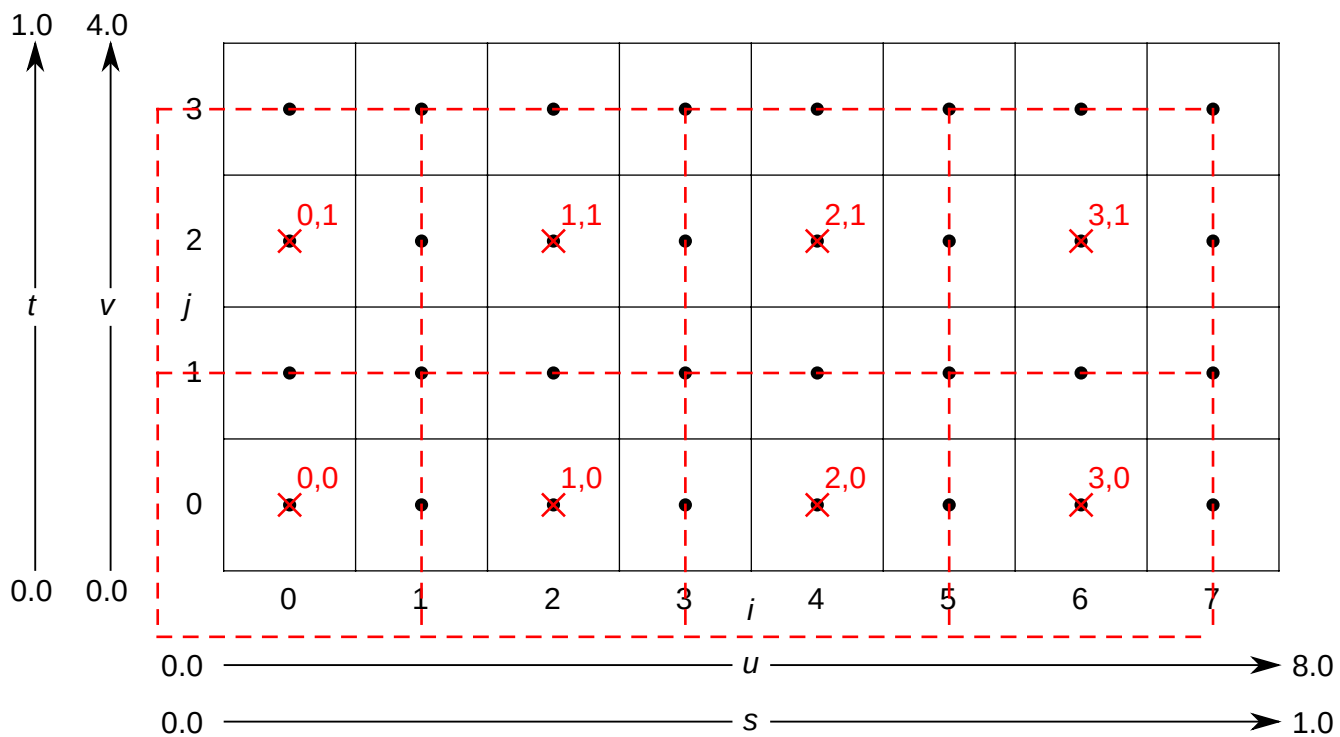


Figure 7. 420 downsampling, $x\text{ChromaOffset}=\text{COSITED\_EVEN}$, $y\text{ChromaOffset}=\text{COSITED\_EVEN}$

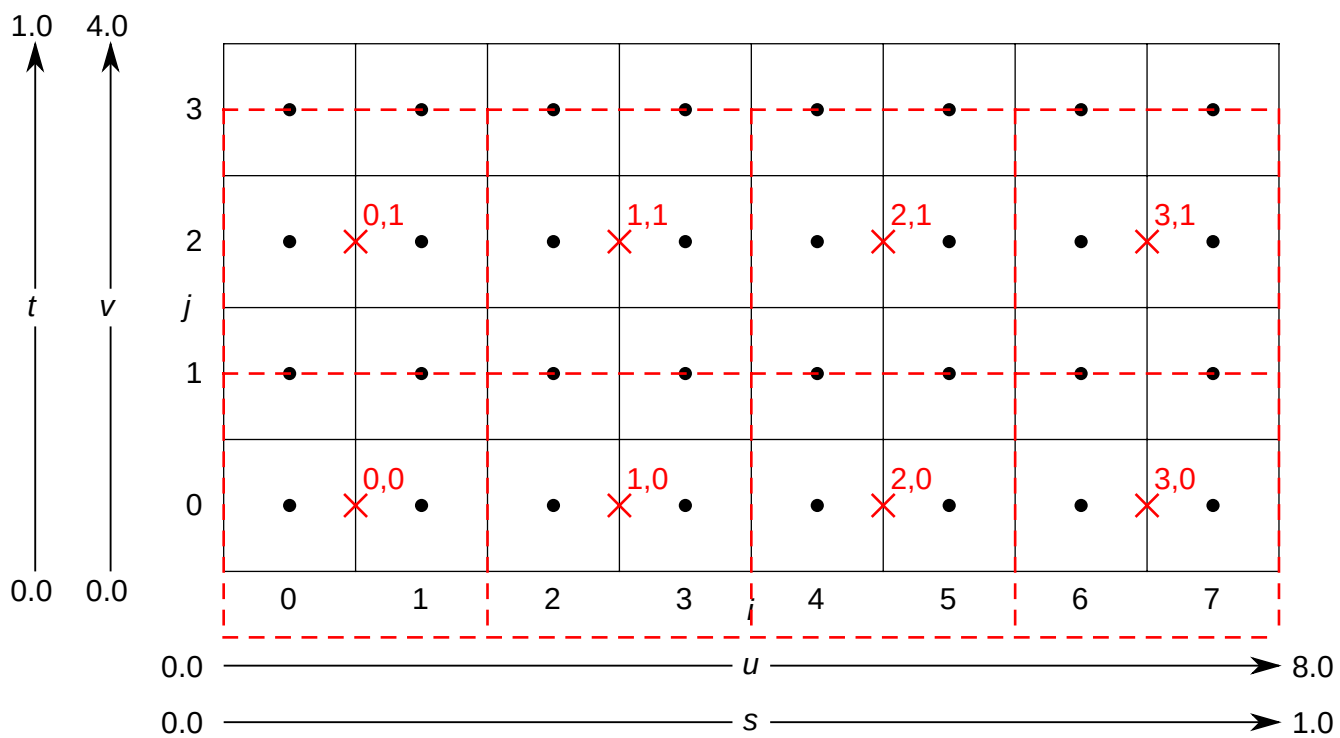


Figure 8. 420 downsampling, $x\text{ChromaOffset}=\text{MIDPOINT}$, $y\text{ChromaOffset}=\text{COSITED\_EVEN}$

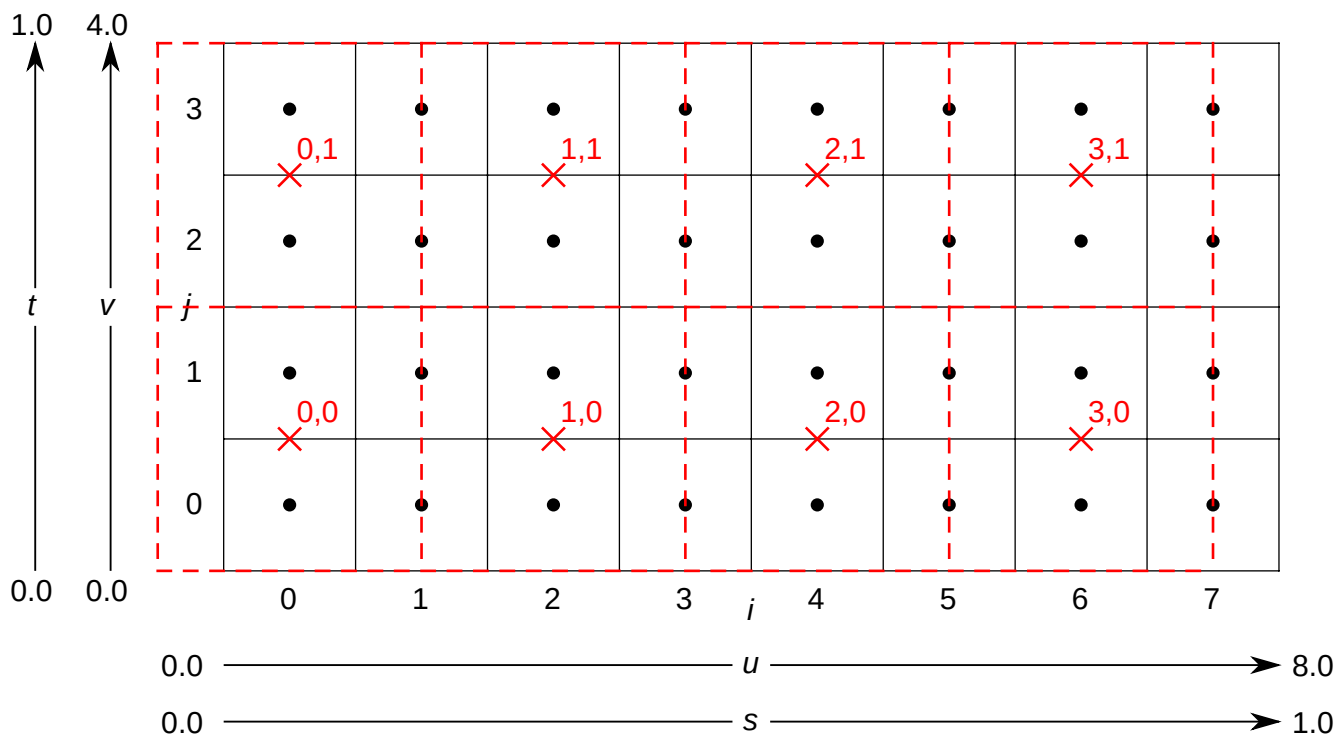


Figure 9. 420 downsampling, $xChromaOffset=COSITED\_EVEN$, $yChromaOffset=MIDPOINT$

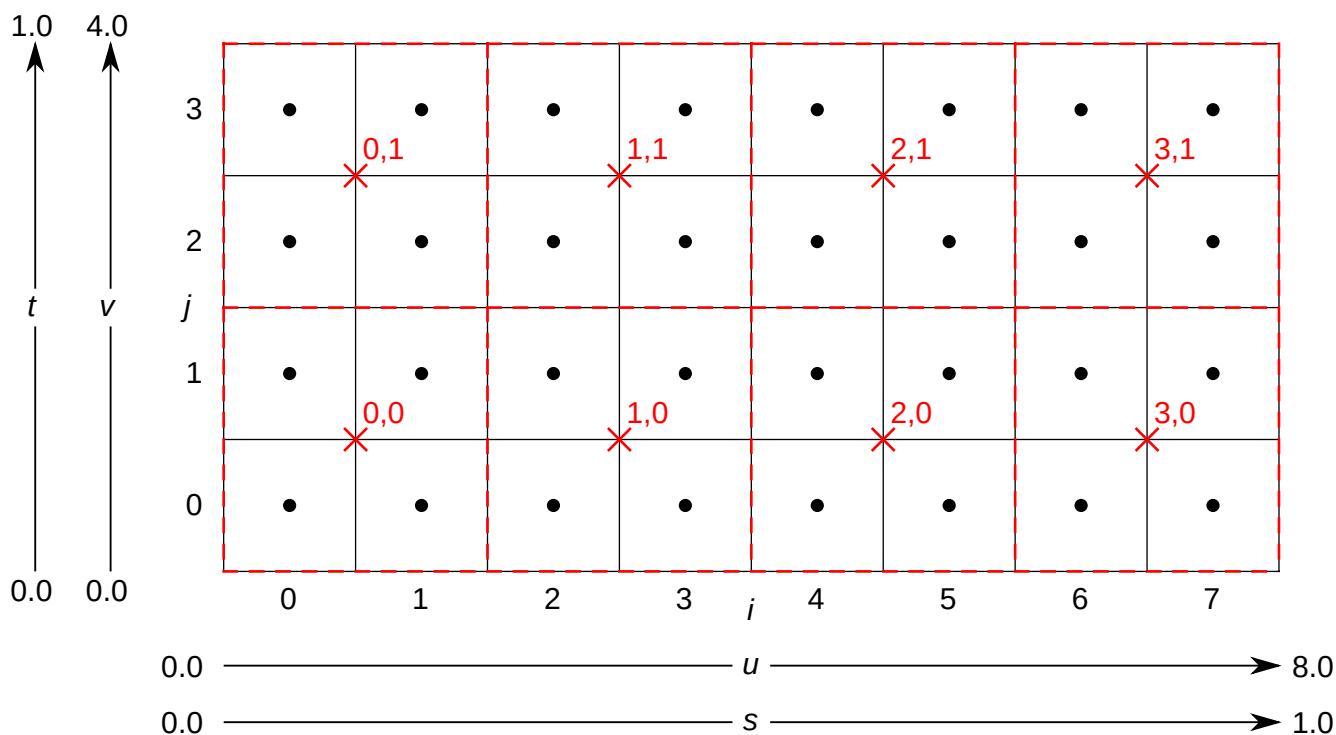


Figure 10. 420 downsampling, $xChromaOffset=MIDPOINT$, $yChromaOffset=MIDPOINT$

Reconstruction is implemented in one of two ways:

If the format of the image that is to be sampled sets `VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT`, or the `VkSamplerYcbcrConversionCreateInfo`'s `forceExplicitReconstruction` is `VK_TRUE`, reconstruction is performed as an explicit step independent of filtering, described in the [Explicit Reconstruction](#) section.

If the format of the image that is to be sampled does not set `VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT` and if the

`VkSamplerYcbcrConversionCreateInfo`'s `forceExplicitReconstruction` is `VK_FALSE`, reconstruction is performed as an implicit part of filtering prior to color model conversion, with no separate post-conversion texel filtering step, as described in the [Implicit Reconstruction](#) section.

Explicit Reconstruction

- If the `chromaFilter` member of the `VkSamplerYcbcrConversionCreateInfo` structure is `VK_FILTER_NEAREST`:
 - If the format's R and B components are reduced in resolution in just width by a factor of two relative to the G component (i.e. this is a “_422” format), the τ_{ijk}^{level} values accessed by [texel filtering](#) are reconstructed as follows:

$$\begin{aligned}\tau_R' [i, j] &= \tau_R [i \times 0.5, j]^{level} \\ \tau_B' [i, j] &= \tau_B [i \times 0.5, j]^{level}\end{aligned}$$

- If the format's R and B components are reduced in resolution in width and height by a factor of two relative to the G component (i.e. this is a “_420” format), the τ_{ijk}^{level} values accessed by [texel filtering](#) are reconstructed as follows:

$$\begin{aligned}\tau_R' [i, j] &= \tau_R [i \times 0.5, \lfloor j \times 0.5 \rfloor]^{level} \\ \tau_B' [i, j] &= \tau_B [i \times 0.5, \lfloor j \times 0.5 \rfloor]^{level}\end{aligned}$$

NOTE

`xChromaOffset` and `yChromaOffset` have no effect if `chromaFilter` is `VK_FILTER_NEAREST` for explicit reconstruction.

- If the `chromaFilter` member of the `VkSamplerYcbcrConversionCreateInfo` structure is `VK_FILTER_LINEAR`:
 - If the format's R and B components are reduced in resolution in just width by a factor of two relative to the G component (i.e. this is a “_422” format):
 - If `xChromaOffset` is `VK_CHROMA_LOCATION_COSITED_EVEN`:

$$\tau_{RB}' [i, j] = \begin{cases} \tau_{RB} [i \times 0.5, j]^{level}, & 0.5 \times i = \lfloor 0.5 \times i \rfloor \\ 0.5 \times \tau_{RB} [i \times 0.5, j]^{level} + \\ 0.5 \times \tau_{RB} [i \times 0.5] + 1, j]^{level}, & 0.5 \times i \neq \lfloor 0.5 \times i \rfloor \end{cases}$$

- If `xChromaOffset` is `VK_CHROMA_LOCATION_MIDPOINT`:

$$\tau_{RB}' [i, j] = \begin{cases} 0.25 \times \tau_{RB} [i \times 0.5] - 1, j]^{level} + \\ 0.75 \times \tau_{RB} [i \times 0.5, j]^{level}, & 0.5 \times i = \lfloor 0.5 \times i \rfloor \\ 0.75 \times \tau_{RB} [i \times 0.5, j]^{level} + \\ 0.25 \times \tau_{RB} [i \times 0.5] + 1, j]^{level}, & 0.5 \times i \neq \lfloor 0.5 \times i \rfloor \end{cases}$$

- If the format's R and B components are reduced in resolution in width and height by a factor of two relative to the G component (i.e. this is a “_420” format), a similar relationship applies. Due to the number of options, these formulae are expressed more concisely as follows:

$$i_{RB} = \begin{cases} 0.5 \times \lceil i \rceil & \text{xChromaOffset=COSITED\_EVEN} \\ 0.5 \times \lceil i \rceil - 0.5 & \text{xChromaOffset=MIDPOINT} \end{cases}$$

$$j_{RB} = \begin{cases} 0.5 \times \lceil j \rceil & \text{yChromaOffset=COSITED\_EVEN} \\ 0.5 \times \lceil j \rceil - 0.5 & \text{yChromaOffset=MIDPOINT} \end{cases}$$

$$i_{floor} = \lfloor i_{RB} \rfloor$$

$$j_{floor} = \lfloor j_{RB} \rfloor$$

$$i_{frac} = i_{RB} - i_{floor}$$

$$j_{frac} = j_{RB} - j_{floor}$$

$$\tau_{RB}(\lceil i \rceil, \lceil j \rceil) = \tau_{RB}(i_{floor}, j_{floor}, \lceil level \rceil) \times (1 - i_{frac}) \times (1 - j_{frac}) +$$

$$\tau_{RB}(1 + i_{floor}, j_{floor}, \lceil level \rceil) \times i_{frac} \times (1 - j_{frac}) +$$

$$\tau_{RB}(i_{floor}, 1 + j_{floor}, \lceil level \rceil) \times (1 - i_{frac}) \times j_{frac} +$$

$$\tau_{RB}(1 + i_{floor}, 1 + j_{floor}, \lceil level \rceil) \times i_{frac} \times j_{frac}$$

NOTE

In the case where the texture itself is bilinearly interpolated as described in [Texel Filtering](#), thus requiring four full-color samples for the filtering operation, and where the reconstruction of these samples uses bilinear interpolation in the chroma components due to `chromaFilter=VK_FILTER_LINEAR`, up to nine chroma samples may be required, depending on the sample location.

Implicit Reconstruction

Implicit reconstruction takes place by the samples being interpolated, as required by the filter settings of the sampler, except that `chromaFilter` takes precedence for the chroma samples.

If `chromaFilter` is `VK_FILTER_NEAREST`, an implementation **may** behave as if `xChromaOffset` and `yChromaOffset` were both `VK_CHROMA_LOCATION_MIDPOINT`, irrespective of the values set.

NOTE

This will not have any visible effect if the locations of the luma samples coincide with the location of the samples used for rasterization.

The sample coordinates are adjusted by the downsample factor of the component (such that, for example, the sample coordinates are divided by two if the component has a downsample factor of two relative to the luma component):

$$u_{RB}(\lceil 422/420 \rceil) = \begin{cases} 0.5 \times \lceil u + 0.5 \rceil, & \text{xChromaOffset=COSITED\_EVEN} \\ 0.5 \times u, & \text{xChromaOffset=MIDPOINT} \end{cases}$$

$$v_{RB}(\lceil 420 \rceil) = \begin{cases} 0.5 \times \lceil v + 0.5 \rceil, & \text{yChromaOffset=COSITED\_EVEN} \\ 0.5 \times v, & \text{yChromaOffset=MIDPOINT} \end{cases}$$

17.2.8. Sampler Y'C_BC_R Conversion

Sampler Y'C_BC_R conversion performs the following operations on sampled data, in order:

1. Sampler Y'C_BC_R Component Swizzle

2. Sampler Y'C_BC_R Range Expansion

3. Sampler Y'C_BC_R Model Conversion

An implementation **may** combine the range expansion and model conversion into a single mathematical operation.

These operations are applied to color component values after [sampling operations](#) which are not specific to sampler Y'C_BC_R conversion have completed.

Sampler Y'C_BC_R Component Swizzle

Rather than the [component swizzle performed as part of sampling](#), which is banned for Y'C_BC_R image views used with sampler Y'C_BC_R conversion, the component swizzle specified by [VkSamplerYcbcrConversionCreateInfo::components](#) is applied to the sampled data instead. This is applied in the same way as the component swizzle usually performed during sampling.

Sampler Y'C_BC_R Range Expansion

Sampler Y'C_BC_R range expansion is not applied if [ycbcrModel](#) is [VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY](#).

For other values of [ycbcrModel](#), range expansion applies independently to each component of the sampled data. For the purposes of range expansion and Y'C_BC_R model conversion, the R and B components contain color difference (chroma) values and the G component contains luma. The A component is not modified by sampler Y'C_BC_R range expansion.

The range expansion to be applied is defined by the [ycbcrRange](#) member of the [VkSamplerYcbcrConversionCreateInfo](#) structure:

- If [ycbcrRange](#) is [VK_SAMPLER_YCBCR_RANGE_ITU_FULL](#), the following transformations are applied:

$$\begin{aligned} Y' &= C'_{rgba} \parallel G_i \\ C_B &= C'_{rgba} \parallel B_i - \frac{2^{in} - 1}{2^n - 1} \\ C_R &= C'_{rgba} \parallel R_i - \frac{2^{in} - 1}{2^n - 1} \end{aligned}$$

NOTE

These formulae correspond to the “full range” encoding in the “Quantization schemes” chapter of the [Khronos Data Format Specification](#).

Should any future amendments be made to the ITU specifications from which these equations are derived, the formulae used by Vulkan **may** also be updated to maintain parity.

- If [ycbcrRange](#) is [VK_SAMPLER_YCBCR_RANGE_ITU_NARROW](#), the following transformations are applied:

$$Y' = \frac{C'_{rgba} \cdot G \times (2^n - 1) - 16 \times 2^{n-8}}{219 \times 2^{n-8}}$$

$$C_B = \frac{C'_{rgba} \cdot B \times (2^n - 1) - 128 \times 2^{n-8}}{224 \times 2^{n-8}}$$

$$C_R = \frac{C'_{rgba} \cdot R \times (2^n - 1) - 128 \times 2^{n-8}}{224 \times 2^{n-8}}$$

NOTE

These formulae correspond to the “narrow range” encoding in the “Quantization schemes” chapter of the [Khronos Data Format Specification](#).

- n is the bit-depth of the components in the format.

The precision of the operations performed during range expansion **must** be at least that of the source format.

An implementation **may** clamp the results of these range expansion operations such that Y' falls in the range $[0,1]$, and/or such that C_B and C_R fall in the range $[-0.5,0.5]$.

Sampler Y'C_BC_R Model Conversion

The range-expanded values are converted between color models, according to the color model conversion specified in the `ycbcrModel` member:

VK_SAMPLER_YCBCR_MODEL_CONVERSION_RGB_IDENTITY

The color components are not modified by the color model conversion since they are assumed already to represent the desired color model in which the shader is operating; Y'C_BC_R range expansion is also ignored.

VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_IDENTITY

The color components are not modified by the color model conversion and are assumed to be treated as though in Y'C_BC_R form both in memory and in the shader; Y'C_BC_R range expansion is applied to the components as for other Y'C_BC_R models, with the vector (C_R,Y',C_B,A) provided to the shader.

VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_709

The color components are transformed from a Y'C_BC_R representation to an R'G'B' representation as described in the “BT.709 Y'C_BC_R conversion” section of the [Khronos Data Format Specification](#).

VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_601

The color components are transformed from a Y'C_BC_R representation to an R'G'B' representation as described in the “BT.601 Y'C_BC_R conversion” section of the [Khronos Data Format Specification](#).

VK_SAMPLER_YCBCR_MODEL_CONVERSION_YCBCR_2020

The color components are transformed from a Y'C_BC_R representation to an R'G'B' representation as described in the “BT.2020 Y'C_BC_R conversion” section of the [Khronos Data Format Specification](#).

In this operation, each output component is dependent on each input component.

An implementation **may** clamp the R'G'B' results of these conversions to the range [0,1].

The precision of the operations performed during model conversion **must** be at least that of the source format.

The alpha component is not modified by these model conversions.

NOTE

Sampling operations in a non-linear color space can introduce color and intensity shifts at sharp transition boundaries. To avoid this issue, the technically precise color correction sequence described in the “Introduction to Color Conversions” chapter of the [Khronos Data Format Specification](#) may be performed as follows:

- Calculate the [unnormalized texel coordinates](#) corresponding to the desired sample position.
- For a [minFilter](#) or [magFilter](#) of [VK_FILTER_NEAREST](#):
 1. Calculate (i,j) for the sample location as described under the “nearest filtering” formulae in [\(u,v,w,a\) to \(i,j,k,l,n\) Transformation and Array Layer Selection](#)
 2. Calculate the normalized texel coordinates corresponding to these integer coordinates.
 3. Sample using [sampler Y'C_BC_R conversion](#) at this location.
- For a [minFilter](#) or [magFilter](#) of [VK_FILTER_LINEAR](#):
 1. Calculate $(i_{[0,1]},j_{[0,1]})$ for the sample location as described under the “linear filtering” formulae in [\(u,v,w,a\) to \(i,j,k,l,n\) Transformation and Array Layer Selection](#)
 2. Calculate the normalized texel coordinates corresponding to these integer coordinates.
 3. Sample using [sampler Y'C_BC_R conversion](#) at each of these locations.
 4. Convert the non-linear A'R'G'B' outputs of the Y'C_BC_R conversions to linear ARGB values as described in the “Transfer Functions” chapter of the [Khronos Data Format Specification](#).
 5. Interpolate the linear ARGB values using the α and β values described in the “linear filtering” section of [\(u,v,w,a\) to \(i,j,k,l,n\) Transformation and Array Layer Selection](#) and the equations in [Texel Filtering](#).

The additional calculations and, especially, additional number of sampling operations in the [VK_FILTER_LINEAR](#) case can be expected to have a performance impact compared with using the outputs directly. Since the variations from “correct” results are subtle for most content, the application author should determine whether a more costly implementation is strictly necessary.

If [chromaFilter](#), and [minFilter](#) or [magFilter](#) are both [VK_FILTER_NEAREST](#), these operations are redundant and sampling using [sampler Y'C_BC_R conversion](#) at the desired sample coordinates will produce the “correct” results without further processing.

17.3. Normalized Texel Coordinate Operations

If the image sampler instruction provides normalized texel coordinates, some of the following operations are performed.

17.3.1. Projection Operation

For **Proj** image operations, the normalized texel coordinates (s,t,r,q,a) and (if present) the D_{ref} coordinate are transformed as follows:

$$\begin{aligned} s &= \frac{s}{q}, & \text{for 1D, 2D, or 3D image} \\ t &= \frac{t}{q}, & \text{for 2D or 3D image} \\ r &= \frac{r}{q}, & \text{for 3D image} \\ D_{ref} &= \frac{D_{ref}}{q}, & \text{if provided} \end{aligned}$$

17.3.2. Derivative Image Operations

Derivatives are used for LOD selection. These derivatives are either implicit (in an **ImplicitLod** image instruction in a fragment shader) or explicit (provided explicitly by shader to the image instruction in any shader).

For implicit derivatives image instructions, the derivatives of texel coordinates are calculated in the same manner as [derivative operations](#). That is:

$$\begin{aligned} \partial s / \partial x &= dPdx \dot{s}, & \partial s / \partial y &= dPdy \dot{s}, & \text{for 1D, 2D, Cube, or 3D image} \\ \partial t / \partial x &= dPdx \dot{t}, & \partial t / \partial y &= dPdy \dot{t}, & \text{for 2D, Cube, or 3D image} \\ \partial r / \partial x &= dPdx \dot{r}, & \partial r / \partial y &= dPdy \dot{r}, & \text{for Cube or 3D image} \end{aligned}$$

Partial derivatives not defined above for certain image dimensionalities are set to zero.

For explicit LOD image instructions, if the **optional** SPIR-V operand **Grad** is provided, then the operand values are used for the derivatives. The number of components present in each derivative for a given image dimensionality matches the number of partial derivatives computed above.

If the **optional** SPIR-V operand **Lod** is provided, then derivatives are set to zero, the cube map derivative transformation is skipped, and the scale factor operation is skipped. Instead, the floating-point scalar coordinate is directly assigned to λ_{base} as described in [LOD Operation](#).

If the image or sampler object used by an implicit derivative image instruction is not uniform across the quad and **quadDivergentImplicitLod** is not supported, then the derivative and LOD values are undefined. Implicit derivatives are well-defined when the image and sampler and control flow are uniform across the quad, even if they diverge between different quads.

If **quadDivergentImplicitLod** is supported, then derivatives and implicit LOD values are well-defined

even if the image or sampler object are not uniform within a quad. The derivatives are computed as specified above, and the implicit LOD calculation proceeds for each shader invocation using its respective image and sampler object.

17.3.3. Cube Map Face Selection and Transformations

For cube map image instructions, the (s,t,r) coordinates are treated as a direction vector (r_x, r_y, r_z). The direction vector is used to select a cube map face. The direction vector is transformed to a per-face texel coordinate system ($s_{\text{face}}, t_{\text{face}}$). The direction vector is also used to transform the derivatives to per-face derivatives.

17.3.4. Cube Map Face Selection

The direction vector selects one of the cube map's faces based on the largest magnitude coordinate direction (the major axis direction). Since two or more coordinates **can** have identical magnitude, the implementation **must** have rules to disambiguate this situation.

The rules **should** have as the first rule that r_z wins over r_y and r_x , and the second rule that r_y wins over r_x . An implementation **may** choose other rules, but the rules **must** be deterministic and depend only on (r_x, r_y, r_z).

The layer number (corresponding to a cube map face), the coordinate selections for s_c , t_c , r_c , and the selection of derivatives, are determined by the major axis direction as specified in the following two tables.

Table 22. Cube Map Face and Coordinate Selection

Major Axis Direction	Layer Number	Cube Map Face	s_c	t_c	r_c
$+r_x$	0	Positive X	$-r_z$	$-r_y$	r_x
$-r_x$	1	Negative X	$+r_z$	$-r_y$	r_x
$+r_y$	2	Positive Y	$+r_x$	$+r_z$	r_y
$-r_y$	3	Negative Y	$+r_x$	$-r_z$	r_y
$+r_z$	4	Positive Z	$+r_x$	$-r_y$	r_z
$-r_z$	5	Negative Z	$-r_x$	$-r_y$	r_z

Table 23. Cube Map Derivative Selection

Major Axis Direction	$\partial s_c / \partial x$	$\partial s_c / \partial y$	$\partial t_c / \partial x$	$\partial t_c / \partial y$	$\partial r_c / \partial x$	$\partial r_c / \partial y$
$+r_x$	$-\partial r_z / \partial x$	$-\partial r_z / \partial y$	$-\partial r_y / \partial x$	$-\partial r_y / \partial y$	$+\partial r_x / \partial x$	$+\partial r_x / \partial y$
$-r_x$	$+\partial r_z / \partial x$	$+\partial r_z / \partial y$	$-\partial r_y / \partial x$	$-\partial r_y / \partial y$	$-\partial r_x / \partial x$	$-\partial r_x / \partial y$
$+r_y$	$+\partial r_x / \partial x$	$+\partial r_x / \partial y$	$+\partial r_z / \partial x$	$+\partial r_z / \partial y$	$+\partial r_y / \partial x$	$+\partial r_y / \partial y$

Major Axis Direction	$\partial s_c / \partial x$	$\partial s_c / \partial y$	$\partial t_c / \partial x$	$\partial t_c / \partial y$	$\partial r_c / \partial x$	$\partial r_c / \partial y$
$-r_y$	$+\partial r_x / \partial x$	$+\partial r_x / \partial y$	$-\partial r_z / \partial x$	$-\partial r_z / \partial y$	$-\partial r_y / \partial x$	$-\partial r_y / \partial y$
$+r_z$	$+\partial r_x / \partial x$	$+\partial r_x / \partial y$	$-\partial r_y / \partial x$	$-\partial r_y / \partial y$	$+\partial r_z / \partial x$	$+\partial r_z / \partial y$
$-r_z$	$-\partial r_x / \partial x$	$-\partial r_x / \partial y$	$-\partial r_y / \partial x$	$-\partial r_y / \partial y$	$-\partial r_z / \partial x$	$-\partial r_z / \partial y$

17.3.5. Cube Map Coordinate Transformation

$$s_{face} = \frac{1}{2} \times \frac{s_c}{|r_c|} + \frac{1}{2}$$

$$t_{face} = \frac{1}{2} \times \frac{t_c}{|r_c|} + \frac{1}{2}$$

17.3.6. Cube Map Derivative Transformation

The partial derivatives of the [Cube Map Coordinate Transformations](#) can be computed as:

$$\begin{aligned} \frac{\partial s_{face}}{\partial x} &= \frac{\partial}{\partial x} \left[\frac{1}{2} \times \frac{s_c}{|r_c|} + \frac{1}{2} \right] \\ &= \frac{1}{2} \times \frac{\partial}{\partial x} \left[\frac{s_c}{|r_c|} \right] \\ &= \frac{1}{2} \times \frac{|r_c| \times \partial s_c / \partial x - s_c \times \partial r_c / \partial x}{|r_c|^2} \end{aligned}$$

The other derivatives are simplified similarly, resulting in

$$\begin{aligned} \frac{\partial s_{face}}{\partial y} &= \frac{1}{2} \times \frac{|r_c| \times \partial s_c / \partial y - s_c \times \partial r_c / \partial y}{|r_c|^2} \\ \frac{\partial t_{face}}{\partial x} &= \frac{1}{2} \times \frac{|r_c| \times \partial t_c / \partial x - t_c \times \partial r_c / \partial x}{|r_c|^2} \\ \frac{\partial t_{face}}{\partial y} &= \frac{1}{2} \times \frac{|r_c| \times \partial t_c / \partial y - t_c \times \partial r_c / \partial y}{|r_c|^2} \end{aligned}$$

17.3.7. Scale Factor Operation, LOD Operation and Image Level(s) Selection

LOD selection **can** be either explicit (provided explicitly by the image instruction) or implicit (determined from a scale factor calculated from the derivatives). The LOD **must** be computed with `mipmapPrecisionBits` of accuracy.

Scale Factor Operation

The magnitude of the derivatives are calculated by:

$$m_{ux} = |\partial s / \partial x| \times w_{base}$$

$$m_{vx} = |\partial t / \partial x| \times h_{base}$$

$$m_{wx} = |\partial r / \partial x| \times d_{base}$$

$$m_{uy} = |\partial s / \partial y| \times w_{base}$$

$$m_{vy} = |\partial t / \partial y| \times h_{base}$$

$$m_{wy} = |\partial r / \partial y| \times d_{base}$$

where:

$$\partial t / \partial x = \partial t / \partial y = 0 \text{ (for 1D images)}$$

$$\partial r / \partial x = \partial r / \partial y = 0 \text{ (for 1D, 2D or Cube images)}$$

and:

$$w_{base} = \text{image.w}$$

$$h_{base} = \text{image.h}$$

$$d_{base} = \text{image.d}$$

(for the `baseMipLevel`, from the image descriptor).

A point sampled in screen space has an elliptical footprint in texture space. The minimum and maximum scale factors (ρ_{min} , ρ_{max}) **should** be the minor and major axes of this ellipse.

The *scale factors* ρ_x and ρ_y , calculated from the magnitude of the derivatives in x and y, are used to compute the minimum and maximum scale factors.

ρ_x and ρ_y **may** be approximated with functions f_x and f_y , subject to the following constraints:

f_x is continuous and monotonically increasing in each of m_{ux} , m_{vx} , and m_{wx}
 f_y is continuous and monotonically increasing in each of m_{uy} , m_{vy} , and m_{wy}

$$\max(|m_{ux}|, |m_{vx}|, |m_{wx}|) \leq f_x \leq \sqrt{2}(|m_{ux}| + |m_{vx}| + |m_{wx}|)$$

$$\max(|m_{uy}|, |m_{vy}|, |m_{wy}|) \leq f_y \leq \sqrt{2}(|m_{uy}| + |m_{vy}| + |m_{wy}|)$$

The minimum and maximum scale factors ($\rho_{\min}, \rho_{\max}$) are determined by:

$$\rho_{\max} = \max(\rho_x, \rho_y)$$

$$\rho_{\min} = \min(\rho_x, \rho_y)$$

The ratio of anisotropy is determined by:

$$\eta = \min(\rho_{\max}/\rho_{\min}, \max_{\text{Aniso}})$$

where:

$\text{sampler.max}_{\text{Aniso}} = \text{maxAnisotropy}$ (from sampler descriptor)

$\text{limits.max}_{\text{Aniso}} = \text{maxSamplerAnisotropy}$ (from physical device limits)

$$\max_{\text{Aniso}} = \min(\text{sampler.max}_{\text{Aniso}}, \text{limits.max}_{\text{Aniso}})$$

If $\rho_{\max} = \rho_{\min} = 0$, then all the partial derivatives are zero, the fragment's footprint in texel space is a point, and η **should** be treated as 1. If $\rho_{\max} \neq 0$ and $\rho_{\min} = 0$ then all partial derivatives along one axis are zero, the fragment's footprint in texel space is a line segment, and η **should** be treated as $\max_{\text{Aniso}}$. However, anytime the footprint is small in texel space the implementation **may** use a smaller value of η , even when $\rho_{\min}$ is zero or close to zero. If either `VkPhysicalDeviceFeatures::samplerAnisotropy` or `VkSamplerCreateInfo::anisotropyEnable` are `VK_FALSE`, $\max_{\text{Aniso}}$ is set to 1.

If $\eta = 1$, sampling is isotropic. If $\eta > 1$, sampling is anisotropic.

The sampling rate (N) is derived as:

$$N = \lceil \eta \rceil$$

An implementation **may** round N up to the nearest supported sampling rate. An implementation **may** use the value of N as an approximation of η .

LOD Operation

The LOD parameter λ is computed as follows:

$$\lambda_{base}(x, y) = \begin{cases} shaderOp.Lod & \text{(from optional SPIR-V operand)} \\ \log_2 \left\lceil \frac{\rho_{max}}{\eta} \right\rceil & \text{otherwise} \end{cases}$$

$$\lambda'(x, y) = \lambda_{base} + \text{clamp}(\text{sampler.bias} + \text{shaderOp.bias}, -\text{maxSamplerLodBias}, \text{maxSamplerLodBias})$$

$$\lambda = \begin{cases} lod_{max}, & \lambda' > lod_{max} \\ \lambda', & lod_{min} \leq \lambda' \leq lod_{max} \\ lod_{min}, & \lambda' < lod_{min} \\ undefined, & lod_{min} > lod_{max} \end{cases}$$

where:

$$\text{sampler.bias} = \text{mipLodBias} \quad \text{(from sampler descriptor)}$$

$$\text{shaderOp.bias} = \begin{cases} Bias & \text{(from optional SPIR-V operand)} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{sampler.lod}_{min} = \text{minLod} \quad \text{(from sampler descriptor)}$$

$$\text{shaderOp.lod}_{min} = \begin{cases} \text{MinLod} & \text{(from optional SPIR-V operand)} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{lod}_{min} = \max(\text{sampler.lod}_{min}, \text{shaderOp.lod}_{min})$$

$$\text{lod}_{max} = \text{maxLod} \quad \text{(from sampler descriptor)}$$

and maxSamplerLodBias is the value of the [VkPhysicalDeviceLimits](#) feature [maxSamplerLodBias](#).

Image Level(s) Selection

The image level(s) d , d_{hi} , and d_{lo} which texels are read from are determined by an image-level parameter d_l , which is computed based on the LOD parameter, as follows:

$$d_l = \begin{cases} \text{nearest}(d'), & \text{mipmapMode is VK\_SAMPLER\_MIPMAP\_MODE\_NEAREST} \\ d', & \text{otherwise} \end{cases}$$

where:

$$d' = \text{level}_{base} + \text{clamp}(\lambda, 0, q)$$

$$\text{nearest}(d') = \begin{cases} \lfloor d' + 0.5 \rfloor - 1, & \text{preferred} \\ \lfloor d' + 0.5 \rfloor, & \text{alternative} \end{cases}$$

and:

$$\text{level}_{base} = \text{baseMipLevel}$$

$$q = \text{levelCount} - 1$$

[baseMipLevel](#) and [levelCount](#) are taken from the [subresourceRange](#) of the image view.

If the sampler's [mipmapMode](#) is [VK_SAMPLER_MIPMAP_MODE_NEAREST](#), then the level selected is $d = d_l$.

If the sampler's [mipmapMode](#) is [VK_SAMPLER_MIPMAP_MODE_LINEAR](#), two neighboring levels are selected:

$$\begin{aligned}
d_{hi} &= \lfloor d_l \rfloor \\
d_{lo} &= \min(d_{hi} + 1, level_{base} + q) \\
\delta &= d_l - d_{hi}
\end{aligned}$$

δ is the fractional value, quantized to the number of [mipmap precision bits](#), used for [linear filtering](#) between levels.

17.3.8. (s,t,r,q,a) to (u,v,w,a) Transformation

The normalized texel coordinates are scaled by the image level dimensions and the array layer is selected.

This transformation is performed once for each level used in [filtering](#) (either d , or d_{hi} and d_{lo}).

$$\begin{aligned}
u[x, y] &= s[x, y] \times width_{scale} + \Delta_i \\
v[x, y] &= \begin{cases} 0 & \text{for 1D images} \\ t[x, y] \times height_{scale} + \Delta_j & \text{otherwise} \end{cases} \\
w[x, y] &= \begin{cases} 0 & \text{for 2D or Cube images} \\ r[x, y] \times depth_{scale} + \Delta_k & \text{otherwise} \end{cases} \\
a[x, y] &= \begin{cases} a[x, y] & \text{for array images} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

where:

$$width_{scale} = width_{level}$$

$$height_{scale} = height_{level}$$

$$depth_{scale} = depth_{level}$$

and where $(\Delta_i, \Delta_j, \Delta_k)$ are taken from the image instruction if it includes a [ConstOffset](#) or [Offset](#) operand, otherwise they are taken to be zero.

Operations then proceed to Unnormalized Texel Coordinate Operations.

17.4. Unnormalized Texel Coordinate Operations

17.4.1. (u,v,w,a) to (i,j,k,l,n) Transformation and Array Layer Selection

The unnormalized texel coordinates are transformed to integer texel coordinates relative to the selected mipmap level.

The layer index l is computed as:

$$l = \text{clamp}(\text{RNE}(a), 0, \text{layerCount} - 1) + \text{baseArrayLayer}$$

where `layerCount` is the number of layers in the image subresource range of the image view, `baseArrayLayer` is the first layer from the subresource range, and where:

$$\text{RNE } \tilde{a} = \begin{cases} \text{roundTiesToEven } \tilde{a} & \text{preferred, from IEEE Std 754-2008 Floating-Point Arithmetic} \\ \lfloor a + 0.5 \rfloor & \text{alternative} \end{cases}$$

The sample index `n` is assigned the value 0.

Nearest filtering (`VK_FILTER_NEAREST`) computes the integer texel coordinates that the unnormalized coordinates lie within:

$$\begin{aligned} i &= \lfloor u + \text{shift} \rfloor \\ j &= \lfloor v + \text{shift} \rfloor \\ k &= \lfloor w + \text{shift} \rfloor \end{aligned}$$

where:

$$\text{shift} = 0.0$$

Linear filtering (`VK_FILTER_LINEAR`) computes a set of neighboring coordinates which bound the unnormalized coordinates. The integer texel coordinates are combinations of i_0 or i_1 , j_0 or j_1 , k_0 or k_1 , as well as weights α , β , and γ .

$$\begin{aligned} i_0 &= \lfloor u - \text{shift} \rfloor \\ i_1 &= i_0 + 1 \\ j_0 &= \lfloor v - \text{shift} \rfloor \\ j_1 &= j_0 + 1 \\ k_0 &= \lfloor w - \text{shift} \rfloor \\ k_1 &= k_0 + 1 \end{aligned}$$

$$\begin{aligned} \alpha &= \text{frac } \tilde{u} - \text{shift} \\ \beta &= \text{frac } \tilde{v} - \text{shift} \\ \gamma &= \text{frac } \tilde{w} - \text{shift} \end{aligned}$$

where:

$$\text{shift} = 0.5$$

and where:

$$\text{frac } \tilde{x} = x - \lfloor x \rfloor$$

where the number of fraction bits retained is specified by `VkPhysicalDeviceLimits::subTexelPrecisionBits`.

Cubic filtering (`VK_FILTER_CUBIC_EXT`) computes a set of neighboring coordinates which bound the unnormalized coordinates. The integer texel coordinates are combinations of i_0, i_1, i_2 or i_3, j_0, j_1, j_2 or j_3, k_0, k_1, k_2 or k_3 , as well as weights α, β , and γ .

$$i_0 = \lfloor u - \frac{3}{2} \rfloor \quad i_1 = i_0 + 1 \quad i_2 = i_1 + 1 \quad i_3 = i_2 + 1$$

$$j_0 = \lfloor v - \frac{3}{2} \rfloor \quad j_1 = j_0 + 1 \quad j_2 = j_1 + 1 \quad j_3 = j_2 + 1$$

$$k_0 = \lfloor w - \frac{3}{2} \rfloor \quad k_1 = k_0 + 1 \quad k_2 = k_1 + 1 \quad k_3 = k_2 + 1$$

$$\alpha = \text{frac} \left(\frac{\lceil u - \frac{1}{2} \rceil}{A} \right)$$

$$\beta = \text{frac} \left(\frac{\lceil v - \frac{1}{2} \rceil}{A} \right)$$

$$\gamma = \text{frac} \left(\frac{\lceil w - \frac{1}{2} \rceil}{A} \right)$$

where:

$$\text{frac} \left(\frac{x}{A} \right) = x - \lfloor x \rfloor$$

where the number of fraction bits retained is specified by `VkPhysicalDeviceLimits::subTexelPrecisionBits`.

17.5. Integer Texel Coordinate Operations

The `OpImageFetch` and `OpImageFetchSparse` SPIR-V instructions **may** supply a LOD from which texels are to be fetched using the optional SPIR-V operand `Lod`. Other integer-coordinate operations **must** not. If the `Lod` is provided then it **must** be an integer.

The image level selected is:

$$d = level_{base} + \begin{cases} Lod & \text{(from optional SPIR-V operand)} \\ 0 & \text{otherwise} \end{cases}$$

If d does not lie in the range $[baseMipLevel, baseMipLevel + levelCount)$ then any values fetched are zero if the `robustImageAccess2` feature is enabled, otherwise are undefined, and any writes (if supported) are discarded.

17.6. Image Sample Operations

17.6.1. Wrapping Operation

`Cube` images ignore the wrap modes specified in the sampler. Instead, if `VK_FILTER_NEAREST` is used

within a mip level then `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE` is used, and if `VK_FILTER_LINEAR` is used within a mip level then sampling at the edges is performed as described earlier in the [Cube map edge handling](#) section.

The first integer texel coordinate i is transformed based on the `addressModeU` parameter of the sampler.

$$i = \begin{cases} i \bmod size & \text{for repeat} \\ size - 1 - \text{mirror } i \bmod 2 \times size - size & \text{for mirrored repeat} \\ \text{clamp } i, 0, size - 1 & \text{for clamp to edge} \\ \text{clamp } i, -1, size & \text{for clamp to border} \\ \text{clamp } \text{mirror } i, 0, size - 1 & \text{for mirror clamp to edge} \end{cases}$$

where:

$$\text{mirror } n = \begin{cases} n & \text{for } n \geq 0 \\ -1 + n & \text{otherwise} \end{cases}$$

j (for 2D and Cube image) and k (for 3D image) are similarly transformed based on the `addressModeV` and `addressModeW` parameters of the sampler, respectively.

17.6.2. Texel Gathering

SPIR-V instructions with `Gather` in the name return a vector derived from 4 texels in the base level of the image view. The rules for the `VK_FILTER_LINEAR` minification filter are applied to identify the four selected texels. Each texel is then converted to an RGBA value according to [component substitution](#) and then [swizzled](#). A four-component vector is then assembled by taking the component indicated by the `Component` value in the instruction from the swizzled color value of the four texels. If the operation does not use the `ConstOffsets` image operand then the four texels form the 2×2 rectangle used for texture filtering:

$$\begin{aligned} \tau[R]_i &= \tau_{i0j1} \text{level}_{base}[i]_{comp}[i] \\ \tau[G]_i &= \tau_{i1j1} \text{level}_{base}[i]_{comp}[i] \\ \tau[B]_i &= \tau_{i1j0} \text{level}_{base}[i]_{comp}[i] \\ \tau[A]_i &= \tau_{i0j0} \text{level}_{base}[i]_{comp}[i] \end{aligned}$$

If the operation does use the `ConstOffsets` image operand then the offsets allow a custom filter to be defined:

$$\begin{aligned} \tau[R]_i &= \tau_{i0j0 + \Delta_0} \text{level}_{base}[i]_{comp}[i] \\ \tau[G]_i &= \tau_{i0j0 + \Delta_1} \text{level}_{base}[i]_{comp}[i] \\ \tau[B]_i &= \tau_{i0j0 + \Delta_2} \text{level}_{base}[i]_{comp}[i] \\ \tau[A]_i &= \tau_{i0j0 + \Delta_3} \text{level}_{base}[i]_{comp}[i] \end{aligned}$$

where:

$$\tau_{\text{level}_{base}}[\text{comp}] = \begin{cases} \tau_{\text{level}_{base}}[R], & \text{for } comp = 0 \\ \tau_{\text{level}_{base}}[G], & \text{for } comp = 1 \\ \tau_{\text{level}_{base}}[B], & \text{for } comp = 2 \\ \tau_{\text{level}_{base}}[A], & \text{for } comp = 3 \end{cases}$$

comp from SPIR-V operand Component

OpImage*Gather must not be used on a sampled image with **sampler Y'C_BC_R conversion** enabled.

17.6.3. Texel Filtering

Texel filtering is first performed for each level (either d or d_{hi} and d_{lo}).

If λ is less than or equal to zero, the texture is said to be *magnified*, and the filter mode within a mip level is selected by the **magFilter** in the sampler. If λ is greater than zero, the texture is said to be *minified*, and the filter mode within a mip level is selected by the **minFilter** in the sampler.

Texel Nearest Filtering

Within a mip level, **VK_FILTER_NEAREST** filtering selects a single value using the (i, j, k) texel coordinates, with all texels taken from layer l.

$$\tau_{\text{level}}[i] = \begin{cases} \tau_{ijk}[\text{level}], & \text{for 3D image} \\ \tau_{ij}[\text{level}], & \text{for 2D or Cube image} \\ \tau_i[\text{level}], & \text{for 1D image} \end{cases}$$

Texel Linear Filtering

Within a mip level, **VK_FILTER_LINEAR** filtering combines 8 (for 3D), 4 (for 2D or Cube), or 2 (for 1D) texel values, together with their linear weights. The linear weights are derived from the fractions computed earlier:

$$\begin{aligned} w_{i_0} &= 1 - \alpha \\ w_{i_1} &= \alpha \\ w_{j_0} &= 1 - \beta \\ w_{j_1} &= \beta \\ w_{k_0} &= 1 - \gamma \\ w_{k_1} &= \gamma \end{aligned}$$

The values of multiple texels, together with their weights, are combined to produce a filtered value.

The **VkSamplerReductionModeCreateInfo::reductionMode** can control the process by which multiple texels, together with their weights, are combined to produce a filtered texture value.

When the **reductionMode** is set (explicitly or implicitly) to **VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE**, a weighted average is computed:

$$\begin{aligned}\tau_{3D} &= \sum_{k=k_0}^{k_1} \sum_{j=j_0}^{j_1} \sum_{i=i_0}^{i_1} w_i w_j w_k \tau_{ijk} \\ \tau_{2D} &= \sum_{j=j_0}^{j_1} \sum_{i=i_0}^{i_1} w_i w_j \tau_{ij} \\ \tau_{1D} &= \sum_{i=i_0}^{i_1} w_i \tau_i\end{aligned}$$

However, if the reduction mode is `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX`, the process operates on the above set of multiple texels, together with their weights, computing a component-wise minimum or maximum, respectively, of the components of the set of texels with non-zero weights.

Texel Cubic Filtering

Within a mip level, `VK_FILTER_CUBIC_EXT`, filtering computes a weighted average of 64 (for 3D), 16 (for 2D), or 4 (for 1D) texel values, together with their Catmull-Rom weights.

Catmull-Rom weights are derived from the fractions computed earlier.

$$\begin{aligned}\begin{bmatrix} w_{i_0} & w_{i_1} & w_{i_2} & w_{i_3} \end{bmatrix} &= \frac{1}{2} \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix} \\ \begin{bmatrix} w_{j_0} & w_{j_1} & w_{j_2} & w_{j_3} \end{bmatrix} &= \frac{1}{2} \begin{bmatrix} 1 & \beta & \beta^2 & \beta^3 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix} \\ \begin{bmatrix} w_{k_0} & w_{k_1} & w_{k_2} & w_{k_3} \end{bmatrix} &= \frac{1}{2} \begin{bmatrix} 1 & \gamma & \gamma^2 & \gamma^3 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix}\end{aligned}$$

The values of multiple texels, together with their weights, are combined to produce a filtered value.

The `VkSamplerReductionModeCreateInfo::reductionMode` can control the process by which multiple texels, together with their weights, are combined to produce a filtered texture value.

When the `reductionMode` is set (explicitly or implicitly) to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, a weighted average is computed:

$$\begin{aligned}\tau_{3D} &= \sum_{k=j_0}^{k_3} \sum_{j=j_0}^{j_3} \sum_{i=i_0}^{i_3} w_i w_j w_k \tau_{ijk} \\ \tau_{2D} &= \sum_{j=j_0}^{j_3} \sum_{i=i_0}^{i_3} w_i w_j \tau_{ij} \\ \tau_{1D} &= \sum_{i=i_0}^{i_3} w_i \tau_i\end{aligned}$$

However, if the reduction mode is `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX`, the process operates on the above set of multiple texels, together with their weights, computing a component-wise minimum or maximum, respectively, of the components of the set of texels with non-zero weights.

Texel Mipmap Filtering

`VK_SAMPLER_MIPMAP_MODE_NEAREST` filtering returns the value of a single mipmap level,

$$\tau = \tau[d].$$

`VK_SAMPLER_MIPMAP_MODE_LINEAR` filtering combines the values of multiple mipmap levels ($\tau[hi]$ and $\tau[lo]$), together with their linear weights.

The linear weights are derived from the fraction computed earlier:

$$\begin{aligned} w_{hi} &= 1 - \delta \\ w_{lo} &= \delta \end{aligned}$$

The values of multiple mipmap levels, together with their weights, are combined to produce a final filtered value.

The `VkSamplerReductionModeCreateInfo::reductionMode` can control the process by which multiple texels, together with their weights, are combined to produce a filtered texture value.

When the `reductionMode` is set (explicitly or implicitly) to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, a weighted average is computed:

$$\tau = w_{hi} \tau[hi] + w_{lo} \tau[lo]$$

However, if the reduction mode is `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX`, the process operates on the above values, together with their weights, computing a component-wise minimum or maximum, respectively, of the components of the values with non-zero weights.

Texel Anisotropic Filtering

Anisotropic filtering is enabled by the `anisotropyEnable` in the sampler. When enabled, the image filtering scheme accounts for a degree of anisotropy.

The particular scheme for anisotropic texture filtering is implementation-dependent. Implementations **should** consider the `magFilter`, `minFilter` and `mipmapMode` of the sampler to control the specifics of the anisotropic filtering scheme used. In addition, implementations **should** consider `minLod` and `maxLod` of the sampler.

NOTE

For historical reasons, vendor implementations of anisotropic filtering interpret these sampler parameters in different ways, particularly in corner cases such as `magFilter`, `minFilter` of `VK_FILTER_NEAREST` or `maxAnisotropy` equal to 1.0. Applications should not expect consistent behavior in such cases, and should use anisotropic filtering only with parameters which are expected to give a quality improvement relative to `LINEAR` filtering.

The following describes one particular approach to implementing anisotropic filtering for the 2D Image case; implementations **may** choose other methods:

Given a `magFilter`, `minFilter` of `VK_FILTER_LINEAR` and a `mipmapMode` of `VK_SAMPLER_MIPMAP_MODE_NEAREST`:

Instead of a single isotropic sample, N isotropic samples are sampled within the image footprint of the image level d to approximate an anisotropic filter. The sum $\tau_{2D_{aniso}}$ is defined using the single isotropic $\tau_{2D}(u,v)$ at level d .

$$\tau_{2D_{aniso}} = \frac{1}{N} \sum_{i=1}^N \tau_{2D} \left(\left\lfloor u - \frac{1}{2} + \frac{i}{N+1}, y \right\rfloor, \left\lfloor v - \frac{1}{2} + \frac{i}{N+1}, y \right\rfloor \right), \quad \text{when } \rho_x > \rho_y$$

$$\tau_{2D_{aniso}} = \frac{1}{N} \sum_{i=1}^N \tau_{2D} \left(\left\lfloor u, y - \frac{1}{2} + \frac{i}{N+1} \right\rfloor, \left\lfloor v, y - \frac{1}{2} + \frac{i}{N+1} \right\rfloor \right), \quad \text{when } \rho_y \geq \rho_x$$

When `VkSamplerReductionModeCreateInfo::reductionMode` is `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, the above summation is used. However, if the reduction mode is `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX`, the process operates on the above values, together with their weights, computing a component-wise minimum or maximum, respectively, of the components of the values with non-zero weights.

17.7. Image Operation Steps

Each step described in this chapter is performed by a subset of the image instructions:

- Texel Input Validation Operations, Format Conversion, Texel Replacement, Conversion to RGBA, and Component Swizzle: Performed by all instructions except `OpImageWrite`.
- Depth Comparison: Performed by `OpImage*Dref` instructions.
- All Texel output operations: Performed by `OpImageWrite`.
- Projection: Performed by all `OpImage*Proj` instructions.
- Derivative Image Operations, Cube Map Operations, Scale Factor Operation, LOD Operation and Image Level(s) Selection, and Texel Anisotropic Filtering: Performed by all `OpImageSample*` and `OpImageSparseSample*` instructions.
- (s,t,r,q,a) to (u,v,w,a) Transformation, Wrapping, and (u,v,w,a) to (i,j,k,l,n) Transformation And Array Layer Selection: Performed by all `OpImageSample`, `OpImageSparseSample`, and `OpImage*Gather` instructions.
- Texel Gathering: Performed by `OpImage*Gather` instructions.
- Texel Filtering: Performed by all `OpImageSample*` and `OpImageSparseSample*` instructions.
- Sparse Residency: Performed by all `OpImageSparse*` instructions.

17.8. Image Query Instructions

17.8.1. Image Property Queries

`OpImageQuerySize`, `OpImageQuerySizeLod`, `OpImageQueryLevels`, and `OpImageQuerySamples` query properties of the image descriptor that would be accessed by a shader image operation. They return 0 if the bound descriptor is a null descriptor.

`OpImageQuerySizeLod` returns the size of the image level identified by the `Level of Detail` operand. If that level does not exist in the image, and the descriptor is not null, then the value returned is undefined.

17.8.2. LOD Query

`OpImageQueryLod` returns the LOD parameters that would be used in an image operation with the given image and coordinates. If the descriptor that would be accessed is a null descriptor then (0,0) is returned. Otherwise, the steps described in this chapter are performed as if for `OpImageSampleImplicitLod`, up to [Scale Factor Operation, LOD Operation and Image Level\(s\) Selection](#). The return value is the vector $(\lambda', d_l - \text{level}_{\text{base}})$. These values **may** be subject to implementation-specific maxima and minima for very large, out-of-range values.

Chapter 18. Queries

Queries provide a mechanism to return information about the processing of a sequence of Vulkan commands. Query operations are asynchronous, and as such, their results are not returned immediately. Instead, their results, and their availability status are stored in a [Query Pool](#). The state of these queries **can** be read back on the host, or copied to a buffer object on the device.

The supported query types are [Occlusion Queries](#), [Pipeline Statistics Queries](#), and [Timestamp Queries](#). [Performance Queries](#) are supported if the associated extension is available.

18.1. Query Pools

Queries are managed using *query pool* objects. Each query pool is a collection of a specific number of queries of a particular type.

Query pools **cannot** be destroyed [\[SCID-4\]](#).

If `VkPhysicalDeviceVulkanSC10Properties::deviceDestroyFreesMemory` is `VK_TRUE`, the memory is returned to the system when the device is destroyed. Otherwise, it **may** not be returned to the system until the process is terminated.

Query pools are represented by `VkQueryPool` handles:

```
// Provided by VK_VERSION_1_0
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkQueryPool)
```

To create a query pool, call:

```
// Provided by VK_VERSION_1_0
VkResult vkCreateQueryPool(
    VkDevice                                device,
    const VkQueryPoolCreateInfo*            pCreateInfo,
    const VkAllocationCallbacks*            pAllocator,
    VkQueryPool*                            pQueryPool);
```

- `device` is the logical device that creates the query pool.
- `pCreateInfo` is a pointer to a [VkQueryPoolCreateInfo](#) structure containing the number and type of queries to be managed by the pool.
- `pAllocator` controls host memory allocation as described in the [Memory Allocation](#) chapter.
- `pQueryPool` is a pointer to a [VkQueryPool](#) handle in which the resulting query pool object is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkCreateQueryPool` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateQueryPool-device-09663
`device` **must** support at least one queue family with one of the `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` capabilities
- VUID-vkCreateQueryPool-device-05068
The number of query pools currently allocated from `device` plus 1 **must** be less than or equal to the total number of query pools requested via `VkDeviceObjectReservationCreateInfo::queryPoolRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateQueryPool-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkCreateQueryPool-pCreateInfo-parameter
`pCreateInfo` **must** be a valid pointer to a valid `VkQueryPoolCreateInfo` structure
- VUID-vkCreateQueryPool-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateQueryPool-pQueryPool-parameter
`pQueryPool` **must** be a valid pointer to a `VkQueryPool` handle
- VUID-vkCreateQueryPool-device-queuecount
The device **must** have been created with at least 1 queue

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkQueryPoolCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkQueryPoolCreateInfo {
    VkStructureType      sType;
    const void*          pNext;
    VkQueryPoolCreateFlags flags;
    VkQueryType          queryType;
    uint32_t             queryCount;
    VkQueryPipelineStatisticFlags pipelineStatistics;
} VkQueryPoolCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkQueryPoolCreateFlagBits`
- `queryType` is a `VkQueryType` value specifying the type of queries managed by the pool.
- `queryCount` is the number of queries managed by the pool.
- `pipelineStatistics` is a bitmask of `VkQueryPipelineStatisticFlagBits` specifying which counters will be returned in queries on the new pool, as described below in [Pipeline Statistics Queries](#).

`pipelineStatistics` is ignored if `queryType` is not `VK_QUERY_TYPE_PIPELINE_STATISTICS`.

Valid Usage

- VUID-VkQueryPoolCreateInfo-queryType-00791
If the `pipelineStatisticsQuery` feature is not enabled, `queryType` **must** not be `VK_QUERY_TYPE_PIPELINE_STATISTICS`
- VUID-VkQueryPoolCreateInfo-queryType-00792
If `queryType` is `VK_QUERY_TYPE_PIPELINE_STATISTICS`, `pipelineStatistics` **must** be a valid combination of `VkQueryPipelineStatisticFlagBits` values
- VUID-VkQueryPoolCreateInfo-queryType-09534
If `queryType` is `VK_QUERY_TYPE_PIPELINE_STATISTICS`, `pipelineStatistics` **must** not be zero
- VUID-VkQueryPoolCreateInfo-queryType-03222
If `queryType` is `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, the `pNext` chain **must** include a `VkQueryPoolPerformanceCreateInfoKHR` structure
- VUID-VkQueryPoolCreateInfo-queryCount-02763
`queryCount` **must** be greater than 0
- VUID-VkQueryPoolCreateInfo-queryType-05046
If `queryType` is `VK_QUERY_TYPE_OCCLUSION` then `queryCount` **must** be less than or equal to the maximum of all `VkDeviceObjectReservationCreateInfo::maxOcclusionQueriesPerPool` values specified when `device` was created
- VUID-VkQueryPoolCreateInfo-queryType-05047
If `queryType` is `VK_QUERY_TYPE_PIPELINE_STATISTICS` then `queryCount` **must** be less than or equal to the maximum of all `VkDeviceObjectReservationCreateInfo::maxPipelineStatisticsQueriesPerPool` values specified when `device` was created

- VUID-VkQueryPoolCreateInfo-queryType-05048
If `queryType` is `VK_QUERY_TYPE_TIMESTAMP` then `queryCount` **must** be less than or equal to the maximum of all `VkDeviceObjectReservationCreateInfo::maxTimestampQueriesPerPool` values specified when `device` was created
- VUID-VkQueryPoolCreateInfo-queryType-05049
If `queryType` is `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR` then `queryCount` **must** be less than or equal to the maximum of all `VkPerformanceQueryReservationInfoKHR::maxPerformanceQueriesPerPool` values specified when `device` was created

Valid Usage (Implicit)

- VUID-VkQueryPoolCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO`
- VUID-VkQueryPoolCreateInfo-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkQueryPoolPerformanceCreateInfoKHR`
- VUID-VkQueryPoolCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkQueryPoolCreateInfo-flags-zeroBitmask
`flags` **must** be 0
- VUID-VkQueryPoolCreateInfo-queryType-parameter
`queryType` **must** be a valid `VkQueryType` value

Bits which **can** be set in `VkQueryPoolCreateInfo::flags`, specifying options for query pools, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkQueryPoolCreateFlagBits {
} VkQueryPoolCreateFlagBits;
```

NOTE

All bits for this type are defined by extensions, and none of those extensions are enabled in this build of the specification.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkQueryPoolCreateFlags;
```

`VkQueryPoolCreateFlags` is a bitmask type for setting a mask of zero or more `VkQueryPoolCreateFlagBits`.

The `VkQueryPoolPerformanceCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkQueryPoolPerformanceCreateInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t            queueFamilyIndex;
    uint32_t            counterIndexCount;
    const uint32_t*     pCounterIndices;
} VkQueryPoolPerformanceCreateInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `queueFamilyIndex` is the queue family index to create this performance query pool for.
- `counterIndexCount` is the length of the `pCounterIndices` array.
- `pCounterIndices` is a pointer to an array of indices into the `vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR::pCounters` to enable in this performance query pool.

Valid Usage

- VUID-VkQueryPoolPerformanceCreateInfoKHR-queueFamilyIndex-03236
`queueFamilyIndex` **must** be a valid queue family index of the device
- VUID-VkQueryPoolPerformanceCreateInfoKHR-performanceCounterQueryPools-03237
The `performanceCounterQueryPools` feature **must** be enabled
- VUID-VkQueryPoolPerformanceCreateInfoKHR-pCounterIndices-03321
Each element of `pCounterIndices` **must** be in the range of counters reported by `vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR` for the queue family specified in `queueFamilyIndex`

Valid Usage (Implicit)

- VUID-VkQueryPoolPerformanceCreateInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_QUERY_POOL_PERFORMANCE_CREATE_INFO_KHR`
- VUID-VkQueryPoolPerformanceCreateInfoKHR-pCounterIndices-parameter
`pCounterIndices` **must** be a valid pointer to an array of `counterIndexCount` `uint32_t` values
- VUID-VkQueryPoolPerformanceCreateInfoKHR-counterIndexCount-arraylength
`counterIndexCount` **must** be greater than 0

To query the number of passes required to query a performance query pool on a physical device, call:

```
// Provided by VK_KHR_performance_query
void vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR(
    VkPhysicalDevice          physicalDevice,
    const VkQueryPoolPerformanceCreateInfoKHR* pPerformanceQueryCreateInfo,
    uint32_t*                 pNumPasses);
```

- `physicalDevice` is the handle to the physical device whose queue family performance query counter properties will be queried.
- `pPerformanceQueryCreateInfo` is a pointer to a `VkQueryPoolPerformanceCreateInfoKHR` of the performance query that is to be created.
- `pNumPasses` is a pointer to an integer related to the number of passes required to query the performance query pool, as described below.

The `pPerformanceQueryCreateInfo` member `VkQueryPoolPerformanceCreateInfoKHR::queueFamilyIndex` **must** be a queue family of `physicalDevice`. The number of passes required to capture the counters specified in the `pPerformanceQueryCreateInfo` member `VkQueryPoolPerformanceCreateInfoKHR::pCounters` is returned in `pNumPasses`.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR-physicalDevice-parameter
`physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR-pPerformanceQueryCreateInfo-parameter
`pPerformanceQueryCreateInfo` **must** be a valid pointer to a valid `VkQueryPoolPerformanceCreateInfoKHR` structure
- VUID-vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR-pNumPasses-parameter
`pNumPasses` **must** be a valid pointer to a `uint32_t` value

Possible values of `VkQueryPoolCreateInfo::queryType`, specifying the type of queries managed by the pool, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkQueryType {
    VK_QUERY_TYPE_OCCLUSION = 0,
    VK_QUERY_TYPE_PIPELINE_STATISTICS = 1,
    VK_QUERY_TYPE_TIMESTAMP = 2,
    // Provided by VK_KHR_performance_query
    VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR = 1000116000,
} VkQueryType;
```

- `VK_QUERY_TYPE_OCCLUSION` specifies an `occlusion query`.

- `VK_QUERY_TYPE_PIPELINE_STATISTICS` specifies a [pipeline statistics query](#).
- `VK_QUERY_TYPE_TIMESTAMP` specifies a [timestamp query](#).
- `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR` specifies a [performance query](#).

18.2. Query Operation

The operation of queries is controlled by the commands [vkCmdBeginQuery](#), [vkCmdEndQuery](#), [vkCmdResetQueryPool](#), [vkCmdCopyQueryPoolResults](#), [vkCmdWriteTimestamp2KHR](#), and [vkCmdWriteTimestamp](#).

In order for a `VkCommandBuffer` to record query management commands, the queue family for which its `VkCommandPool` was created **must** support the appropriate type of operations (graphics, compute) suitable for the query type of a given query pool.

Each query in a query pool has a status that is either *unavailable* or *available*, and also has state to store the numerical results of a query operation of the type requested when the query pool was created. Resetting a query via [vkCmdResetQueryPool](#) or [vkResetQueryPool](#) sets the status to unavailable and makes the numerical results undefined. A query is made available by the operation of [vkCmdEndQuery](#), [vkCmdWriteTimestamp2KHR](#), or [vkCmdWriteTimestamp](#). Both the availability status and numerical results **can** be retrieved by calling either [vkGetQueryPoolResults](#) or [vkCmdCopyQueryPoolResults](#).

After query pool creation, each query is in an uninitialized state and **must** be reset before it is used. Queries **must** also be reset between uses.

If a logical device includes multiple physical devices, then each command that writes a query **must** execute on a single physical device, and any call to [vkCmdBeginQuery](#) **must** execute the corresponding [vkCmdEndQuery](#) command on the same physical device.

To reset a range of queries in a query pool on a queue, call:

```
// Provided by VK_VERSION_1_0
void vkCmdResetQueryPool(
    VkCommandBuffer          commandBuffer,
    VkQueryPool              queryPool,
    uint32_t                 firstQuery,
    uint32_t                 queryCount);
```

- `commandBuffer` is the command buffer into which this command will be recorded.
- `queryPool` is the handle of the query pool managing the queries being reset.
- `firstQuery` is the initial query index to reset.
- `queryCount` is the number of queries to reset.

When executed on a queue, this command sets the status of query indices [`firstQuery`, `firstQuery` + `queryCount` - 1] to unavailable.

This command defines an execution dependency between other query commands that reference

the same query.

The first [synchronization scope](#) includes all commands which reference the queries in [queryPool](#) indicated by [firstQuery](#) and [queryCount](#) that occur earlier in [submission order](#).

The second [synchronization scope](#) includes all commands which reference the queries in [queryPool](#) indicated by [firstQuery](#) and [queryCount](#) that occur later in [submission order](#).

The operation of this command happens after the first scope and happens before the second scope.

If the [queryType](#) used to create [queryPool](#) was [VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR](#), this command sets the status of query indices [[firstQuery](#), [firstQuery](#) + [queryCount](#) - 1] to unavailable for each pass of [queryPool](#), as indicated by a call to [vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR](#).

NOTE

Because [vkCmdResetQueryPool](#) resets all the passes of the indicated queries, applications must not record a [vkCmdResetQueryPool](#) command for a [queryPool](#) created with [VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR](#) in a command buffer that needs to be submitted multiple times as indicated by a call to [vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR](#). Otherwise applications will never be able to complete the recorded queries.

Valid Usage

- VUID-vkCmdResetQueryPool-firstQuery-09436
[firstQuery](#) **must** be less than the number of queries in [queryPool](#)
- VUID-vkCmdResetQueryPool-firstQuery-09437
The sum of [firstQuery](#) and [queryCount](#) **must** be less than or equal to the number of queries in [queryPool](#)
- VUID-vkCmdResetQueryPool-None-02841
All queries used by the command **must** not be active
- VUID-vkCmdResetQueryPool-firstQuery-02862
If [queryPool](#) was created with [VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR](#), this command **must** not be recorded in a command buffer that, either directly or through secondary command buffers, also contains begin commands for a query from the set of queries [[firstQuery](#), [firstQuery](#) + [queryCount](#) - 1]

Valid Usage (Implicit)

- VUID-vkCmdResetQueryPool-commandBuffer-parameter
[commandBuffer](#) **must** be a valid [VkCommandBuffer](#) handle
- VUID-vkCmdResetQueryPool-queryPool-parameter
[queryPool](#) **must** be a valid [VkQueryPool](#) handle
- VUID-vkCmdResetQueryPool-commandBuffer-recording
[commandBuffer](#) **must** be in the [recording state](#)

- VUID-vkCmdResetQueryPool-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdResetQueryPool-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdResetQueryPool-commonparent
Both of `commandBuffer`, and `queryPool` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT	Action

To reset a range of queries in a query pool on the host, call:

```
// Provided by VK_VERSION_1_2
void vkResetQueryPool(
    VkDevice          device,
    VkQueryPool       queryPool,
    uint32_t          firstQuery,
    uint32_t          queryCount);
```

- `device` is the logical device that owns the query pool.
- `queryPool` is the handle of the query pool managing the queries being reset.
- `firstQuery` is the initial query index to reset.
- `queryCount` is the number of queries to reset.

This command sets the status of query indices [`firstQuery`, `firstQuery + queryCount - 1`] to unavailable.

If `queryPool` is `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR` this command sets the status of query indices

[[firstQuery](#), [firstQuery](#) + [queryCount](#) - 1] to unavailable for each pass.

Valid Usage

- VUID-vkResetQueryPool-firstQuery-09436
[firstQuery](#) **must** be less than the number of queries in [queryPool](#)
- VUID-vkResetQueryPool-firstQuery-09437
The sum of [firstQuery](#) and [queryCount](#) **must** be less than or equal to the number of queries in [queryPool](#)
- VUID-vkResetQueryPool-None-02665
The [hostQueryReset](#) feature **must** be enabled
- VUID-vkResetQueryPool-firstQuery-02741
Submitted commands that refer to the range specified by [firstQuery](#) and [queryCount](#) in [queryPool](#) **must** have completed execution
- VUID-vkResetQueryPool-firstQuery-02742
The range of queries specified by [firstQuery](#) and [queryCount](#) in [queryPool](#) **must** not be in use by calls to [vkGetQueryPoolResults](#) or [vkResetQueryPool](#) in other threads

Valid Usage (Implicit)

- VUID-vkResetQueryPool-device-parameter
[device](#) **must** be a valid [VkDevice](#) handle
- VUID-vkResetQueryPool-queryPool-parameter
[queryPool](#) **must** be a valid [VkQueryPool](#) handle
- VUID-vkResetQueryPool-queryPool-parent
[queryPool](#) **must** have been created, allocated, or retrieved from [device](#)

Once queries are reset and ready for use, query commands **can** be issued to a command buffer. Occlusion queries and pipeline statistics queries count events - drawn samples and pipeline stage invocations, respectively - resulting from commands that are recorded between a [vkCmdBeginQuery](#) command and a [vkCmdEndQuery](#) command within a specified command buffer, effectively scoping a set of drawing and/or dispatching commands. Timestamp queries write timestamps to a query pool. Performance queries record performance counters to a query pool.

A query **must** begin and end in the same command buffer, although if it is a primary command buffer, and the [inheritedQueries](#) feature is enabled, it **can** execute secondary command buffers during the query operation. For a secondary command buffer to be executed while a query is active, it **must** set the [occlusionQueryEnable](#), [queryFlags](#), and/or [pipelineStatistics](#) members of [VkCommandBufferInheritanceInfo](#) to conservative values, as described in the [Command Buffer Recording](#) section. A query **must** either begin and end inside the same subpass of a render pass instance, or **must** both begin and end outside of a render pass instance (i.e. contain entire render pass instances).

If queries are used while executing a render pass instance that has multiview enabled, the query uses N consecutive query indices in the query pool (starting at `query`) where N is the number of bits set in the view mask in the subpass the query is used in. How the numerical results of the query are distributed among the queries is implementation-dependent. For example, some implementations **may** write each view's results to a distinct query, while other implementations **may** write the total result to the first query and write zero to the other queries. However, the sum of the results in all the queries **must** accurately reflect the total result of the query summed over all views. Applications **can** sum the results from all the queries to compute the total result.

Queries used with multiview rendering **must** not span subpasses, i.e. they **must** begin and end in the same subpass.

To begin a query, call:

```
// Provided by VK_VERSION_1_0
void vkCmdBeginQuery(
    VkCommandBuffer          commandBuffer,
    VkQueryPool              queryPool,
    uint32_t                 query,
    VkQueryControlFlags      flags);
```

- `commandBuffer` is the command buffer into which this command will be recorded.
- `queryPool` is the query pool that will manage the results of the query.
- `query` is the query index within the query pool that will contain the results.
- `flags` is a bitmask of `VkQueryControlFlagBits` specifying constraints on the types of queries that **can** be performed.

If the `queryType` of the pool is `VK_QUERY_TYPE_OCCLUSION` and `flags` contains `VK_QUERY_CONTROL_PRECISE_BIT`, an implementation **must** return a result that matches the actual number of samples passed. This is described in more detail in [Occlusion Queries](#).

After beginning a query, that query is considered *active* within the command buffer it was called in until that same query is ended. Queries active in a primary command buffer when secondary command buffers are executed are considered active for those secondary command buffers.

This command defines an execution dependency between other query commands that reference the same query.

The first [synchronization scope](#) includes all commands which reference the queries in `queryPool` indicated by `query` that occur earlier in [submission order](#).

The second [synchronization scope](#) includes all commands which reference the queries in `queryPool` indicated by `query` that occur later in [submission order](#).

The operation of this command happens after the first scope and happens before the second scope.

Valid Usage

- VUID-vkCmdBeginQuery-None-00807
All queries used by the command **must** be *unavailable*
- VUID-vkCmdBeginQuery-queryType-02804
The `queryType` used to create `queryPool` **must** not be `VK_QUERY_TYPE_TIMESTAMP`
- VUID-vkCmdBeginQuery-queryType-00800
If the `occlusionQueryPrecise` feature is not enabled, or the `queryType` used to create `queryPool` was not `VK_QUERY_TYPE_OCCLUSION`, `flags` **must** not contain `VK_QUERY_CONTROL_PRECISE_BIT`
- VUID-vkCmdBeginQuery-query-00802
`query` **must** be less than the number of queries in `queryPool`
- VUID-vkCmdBeginQuery-queryType-00803
If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_OCCLUSION`, the `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdBeginQuery-queryType-00804
If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_PIPELINE_STATISTICS` and any of the `pipelineStatistics` indicate graphics operations, the `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- VUID-vkCmdBeginQuery-queryType-00805
If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_PIPELINE_STATISTICS` and any of the `pipelineStatistics` indicate compute operations, the `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- VUID-vkCmdBeginQuery-commandBuffer-01885
`commandBuffer` **must** not be a protected command buffer
- VUID-vkCmdBeginQuery-query-00808
If called within a render pass instance, the sum of `query` and the number of bits set in the current subpass's view mask **must** be less than or equal to the number of queries in `queryPool`
- VUID-vkCmdBeginQuery-queryPool-01922
`queryPool` **must** have been created with a `queryType` that differs from that of any queries that are `active` within `commandBuffer`
- VUID-vkCmdBeginQuery-queryPool-07289
If `queryPool` was created with a `queryType` of `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, then the `VkQueryPoolPerformanceCreateInfoKHR::queueFamilyIndex` `queryPool` was created with **must** equal the queue family index of the `VkCommandPool` that `commandBuffer` was allocated from
- VUID-vkCmdBeginQuery-queryPool-03223
If `queryPool` was created with a `queryType` of `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, the `profiling lock` **must** have been held before `vkBeginCommandBuffer` was called on `commandBuffer`
- VUID-vkCmdBeginQuery-queryPool-03224

If `queryPool` was created with a `queryType` of `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR` and one of the counters used to create `queryPool` was `VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_BUFFER_KHR`, the query begin **must** be the first recorded command in `commandBuffer`

- VUID-vkCmdBeginQuery-queryPool-03225

If `queryPool` was created with a `queryType` of `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR` and one of the counters used to create `queryPool` was `VK_PERFORMANCE_COUNTER_SCOPE_RENDER_PASS_KHR`, the begin command **must** not be recorded within a render pass instance

- VUID-vkCmdBeginQuery-queryPool-03226

If `queryPool` was created with a `queryType` of `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR` and another query pool with a `queryType` `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR` has been used within `commandBuffer`, its parent primary command buffer or secondary command buffer recorded within the same parent primary command buffer as `commandBuffer`, the `performanceCounterMultipleQueryPools` feature **must** be enabled

- VUID-vkCmdBeginQuery-None-02863

If `queryPool` was created with a `queryType` of `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, this command **must** not be recorded in a command buffer that, either directly or through secondary command buffers, also contains a `vkCmdResetQueryPool` command affecting the same query

Valid Usage (Implicit)

- VUID-vkCmdBeginQuery-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle

- VUID-vkCmdBeginQuery-queryPool-parameter
`queryPool` **must** be a valid `VkQueryPool` handle

- VUID-vkCmdBeginQuery-flags-parameter
`flags` **must** be a valid combination of `VkQueryControlFlagBits` values

- VUID-vkCmdBeginQuery-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state

- VUID-vkCmdBeginQuery-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` operations

- VUID-vkCmdBeginQuery-commonparent
Both of `commandBuffer`, and `queryPool` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally

synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT	Action State

Bits which **can** be set in `vkCmdBeginQuery::flags`, specifying constraints on the types of queries that **can** be performed, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkQueryControlFlagBits {
    VK_QUERY_CONTROL_PRECISE_BIT = 0x00000001,
} VkQueryControlFlagBits;
```

- `VK_QUERY_CONTROL_PRECISE_BIT` specifies the precision of [occlusion queries](#).

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkQueryControlFlags;
```

`VkQueryControlFlags` is a bitmask type for setting a mask of zero or more `VkQueryControlFlagBits`.

To end a query after the set of desired drawing or dispatching commands is executed, call:

```
// Provided by VK_VERSION_1_0
void vkCmdEndQuery(
    VkCommandBuffer          commandBuffer,
    VkQueryPool              queryPool,
    uint32_t                 query);
```

- `commandBuffer` is the command buffer into which this command will be recorded.
- `queryPool` is the query pool that is managing the results of the query.
- `query` is the query index within the query pool where the result is stored.

The command completes the query in `queryPool` identified by `query`, and marks it as available.

This command defines an execution dependency between other query commands that reference the same query.

The first [synchronization scope](#) includes all commands which reference the queries in [queryPool](#) indicated by [query](#) that occur earlier in [submission order](#).

The second [synchronization scope](#) includes only the operation of this command.

Valid Usage

- VUID-vkCmdEndQuery-None-01923
All queries used by the command **must** be [active](#)
- VUID-vkCmdEndQuery-query-00810
[query](#) **must** be less than the number of queries in [queryPool](#)
- VUID-vkCmdEndQuery-commandBuffer-01886
[commandBuffer](#) **must** not be a protected command buffer
- VUID-vkCmdEndQuery-query-00812
If [vkCmdEndQuery](#) is called within a render pass instance, the sum of [query](#) and the number of bits set in the current subpass's view mask **must** be less than or equal to the number of queries in [queryPool](#)
- VUID-vkCmdEndQuery-queryPool-03227
If [queryPool](#) was created with a [queryType](#) of [VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR](#) and one or more of the counters used to create [queryPool](#) was [VK_PERFORMANCE_COUNTER_SCOPE_COMMAND_BUFFER_KHR](#), the [vkCmdEndQuery](#) **must** be the last recorded command in [commandBuffer](#)
- VUID-vkCmdEndQuery-queryPool-03228
If [queryPool](#) was created with a [queryType](#) of [VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR](#) and one or more of the counters used to create [queryPool](#) was [VK_PERFORMANCE_COUNTER_SCOPE_RENDER_PASS_KHR](#), the [vkCmdEndQuery](#) **must** not be recorded within a render pass instance
- VUID-vkCmdEndQuery-None-07007
If called within a subpass of a render pass instance, the corresponding [vkCmdBeginQuery*](#) command **must** have been called previously within the same subpass

Valid Usage (Implicit)

- VUID-vkCmdEndQuery-commandBuffer-parameter
[commandBuffer](#) **must** be a valid [VkCommandBuffer](#) handle
- VUID-vkCmdEndQuery-queryPool-parameter
[queryPool](#) **must** be a valid [VkQueryPool](#) handle
- VUID-vkCmdEndQuery-commandBuffer-recording
[commandBuffer](#) **must** be in the [recording state](#)
- VUID-vkCmdEndQuery-commandBuffer-cmdpool
The [VkCommandPool](#) that [commandBuffer](#) was allocated from **must** support [VK_QUEUE_COMPUTE_BIT](#), or [VK_QUEUE_GRAPHICS_BIT](#) operations

- VUID-vkCmdEndQuery-commonparent

Both of **commandBuffer**, and **queryPool** **must** have been created, allocated, or retrieved from the same **VkDevice**

Host Synchronization

- Host access to **commandBuffer** **must** be externally synchronized
- Host access to the **VkCommandPool** that **commandBuffer** was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHIC_BIT	Action State

An application **can** retrieve results either by requesting they be written into application-provided memory, or by requesting they be copied into a **VkBuffer**. In either case, the layout in memory is defined as follows:

- The first query's result is written starting at the first byte requested by the command, and each subsequent query's result begins **stride** bytes later.
- Each query writes results in a tightly packed array of result values as follows:
 - Occlusion queries write one unsigned integer value - the number of samples passed.
 - Pipeline statistics queries write one unsigned integer value for each bit that is enabled in the **pipelineStatistics** when the pool is created, and the statistics values are written in bit order starting from the least significant bit.
 - Timestamp queries write one unsigned integer value.
 - Performance queries write one **VkPerformanceCounterResultKHR** value for each **VkPerformanceCounterKHR** in the query.
- If **VK_QUERY_RESULT_WITH_AVAILABILITY_BIT** is specified, an additional unsigned integer result value is written indicating whether the query's result is available, with any non-zero value indicating that it is available.
- The bit width of all integer result values written by all query types is either 32- or 64-bits, as requested by the command. The only exceptions are the **VkPerformanceCounterResultKHR** values written by performance queries whose type is determined by the **unit** member of the corresponding **VkPerformanceCounterKHR**.
- If more than one query is retrieved and **stride** is not at least as large as the size of the array of

values corresponding to a single query, the values written to memory are undefined.

To retrieve status and results for a set of queries, call:

```
// Provided by VK_VERSION_1_0
VkResult vkGetQueryPoolResults(
    VkDevice                device,
    VkQueryPool             queryPool,
    uint32_t                firstQuery,
    uint32_t                queryCount,
    size_t                  dataSize,
    void*                   pData,
    VkDeviceSize            stride,
    VkQueryResultFlags      flags);
```

- **device** is the logical device that owns the query pool.
- **queryPool** is the query pool managing the queries containing the desired results.
- **firstQuery** is the initial query index.
- **queryCount** is the number of queries to read.
- **dataSize** is the size in bytes of the buffer pointed to by **pData**.
- **pData** is a pointer to an application-allocated buffer where the results will be written
- **stride** is the stride in bytes between results for individual queries within **pData**.
- **flags** is a bitmask of **VkQueryResultFlagBits** specifying how and when results are returned.

Any results written for a query are written according to [a layout dependent on the query type](#).

If no bits are set in **flags**, and all requested queries are in the available state, results are written as an array of 32-bit unsigned integer values. Behavior when not all queries are available is described [below](#).

If **VK_QUERY_RESULT_WITH_AVAILABILITY_BIT** is set, results for all queries in **queryPool** identified by **firstQuery** and **queryCount** are copied to **pData**, along with an extra availability value written directly after the results of each query and interpreted as an unsigned integer. A value of zero indicates that the results are not yet available, otherwise the query is complete and results are available. The size of the availability values is 64 bits if **VK_QUERY_RESULT_64_BIT** is set in **flags**. Otherwise, it is 32 bits.

NOTE

If **VK_QUERY_RESULT_WITH_AVAILABILITY_BIT** is set, the layout of data in the buffer is a *(result,availability)* pair for each query returned, and **stride** is the stride between each pair.

Results for any available query written by this command are final and represent the final result of the query. If **VK_QUERY_RESULT_PARTIAL_BIT** is set, then for any query that is unavailable, an intermediate result between zero and the final result value is written for that query. Otherwise, any result written by this command is undefined.

If `VK_QUERY_RESULT_64_BIT` is set, results and, if returned, availability values for all queries are written as an array of 64-bit values. If the `queryPool` was created with `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, results for each query are written as an array of the type indicated by `VkPerformanceCounterKHR::storage` for the counter being queried. Otherwise, results and availability values are written as an array of 32-bit values. If an unsigned integer query's value overflows the result type, the value **may** either wrap or saturate. If a signed integer query's value overflows the result type, the value is undefined. If a floating-point query's value is not representable as the result type, the value is undefined.

If `VK_QUERY_RESULT_WAIT_BIT` is set, this command defines an execution dependency with any earlier commands that writes one of the identified queries. The first `synchronization scope` includes all instances of `vkCmdEndQuery`, `vkCmdWriteTimestamp2KHR`, and `vkCmdWriteTimestamp` that reference any query in `queryPool` indicated by `firstQuery` and `queryCount`. The second `synchronization scope` includes the host operations of this command.

If `VK_QUERY_RESULT_WAIT_BIT` is not set, `vkGetQueryPoolResults` **may** return `VK_NOT_READY` if there are queries in the unavailable state.

NOTE

Applications **must** take care to ensure that use of the `VK_QUERY_RESULT_WAIT_BIT` bit has the desired effect.

For example, if a query has been used previously and a command buffer records the commands `vkCmdResetQueryPool`, `vkCmdBeginQuery`, and `vkCmdEndQuery` for that query, then the query will remain in the available state until `vkResetQueryPool` is called or the `vkCmdResetQueryPool` command executes on a queue. Applications **can** use fences or events to ensure that a query has already been reset before checking for its results or availability status. Otherwise, a stale value could be returned from a previous use of the query.

The above also applies when `VK_QUERY_RESULT_WAIT_BIT` is used in combination with `VK_QUERY_RESULT_WITH_AVAILABILITY_BIT`. In this case, the returned availability status **may** reflect the result of a previous use of the query unless `vkResetQueryPool` is called or the `vkCmdResetQueryPool` command has been executed since the last use of the query.

NOTE

Applications **can** double-buffer query pool usage, with a pool per frame, and reset queries at the end of the frame in which they are read.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetQueryPoolResults` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetQueryPoolResults-firstQuery-09436
`firstQuery` **must** be less than the number of queries in `queryPool`
- VUID-vkGetQueryPoolResults-firstQuery-09437
The sum of `firstQuery` and `queryCount` **must** be less than or equal to the number of queries in `queryPool`

- VUID-vkGetQueryPoolResults-queryCount-09438
If **queryCount** is greater than 1, **stride** **must** not be zero
- VUID-vkGetQueryPoolResults-queryType-09439
If the **queryType** used to create **queryPool** was **VK_QUERY_TYPE_TIMESTAMP**, **flags** **must** not contain **VK_QUERY_RESULT_PARTIAL_BIT**
- VUID-vkGetQueryPoolResults-queryType-09440
If the **queryType** used to create **queryPool** was **VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR**, **flags** **must** not contain **VK_QUERY_RESULT_WITH_AVAILABILITY_BIT**, **VK_QUERY_RESULT_PARTIAL_BIT**, or **VK_QUERY_RESULT_64_BIT**
- VUID-vkGetQueryPoolResults-queryType-09441
If the **queryType** used to create **queryPool** was **VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR**, the **queryPool** **must** have been recorded once for each pass as retrieved via a call to [vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR](#)
- VUID-vkGetQueryPoolResults-None-09401
All queries used by the command **must** not be uninitialized
- VUID-vkGetQueryPoolResults-flags-02828
If **VK_QUERY_RESULT_64_BIT** is not set in **flags** and the **queryType** used to create **queryPool** was not **VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR**, then **pData** and **stride** **must** be multiples of 4
- VUID-vkGetQueryPoolResults-flags-00815
If **VK_QUERY_RESULT_64_BIT** is set in **flags** then **pData** and **stride** **must** be multiples of 8
- VUID-vkGetQueryPoolResults-stride-08993
If **VK_QUERY_RESULT_WITH_AVAILABILITY_BIT** is set, **stride** **must** be large enough to contain the unsigned integer representing availability in addition to the query result
- VUID-vkGetQueryPoolResults-queryType-03229
If the **queryType** used to create **queryPool** was **VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR**, then **pData** and **stride** **must** be multiples of the size of [VkPerformanceCounterResultKHR](#)
- VUID-vkGetQueryPoolResults-queryType-04519
If the **queryType** used to create **queryPool** was **VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR**, then **stride** **must** be large enough to contain the [VkQueryPoolPerformanceCreateInfoKHR::counterIndexCount](#) used to create **queryPool** times the size of [VkPerformanceCounterResultKHR](#)
- VUID-vkGetQueryPoolResults-dataSize-00817
dataSize **must** be large enough to contain the result of each query, as described [here](#)

Valid Usage (Implicit)

- VUID-vkGetQueryPoolResults-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkGetQueryPoolResults-queryPool-parameter
queryPool **must** be a valid [VkQueryPool](#) handle
- VUID-vkGetQueryPoolResults-pData-parameter
pData **must** be a valid pointer to an array of **dataSize** bytes

- VUID-vkGetQueryPoolResults-flags-parameter
flags **must** be a valid combination of `VkQueryResultFlagBits` values
- VUID-vkGetQueryPoolResults-dataSize-arraylength
dataSize **must** be greater than 0
- VUID-vkGetQueryPoolResults-queryPool-parent
queryPool **must** have been created, allocated, or retrieved from **device**

Return Codes

Success

- `VK_NOT_READY`
- `VK_SUCCESS`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

Bits which **can** be set in `vkGetQueryPoolResults::flags` and `vkCmdCopyQueryPoolResults::flags`, specifying how and when results are returned, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkQueryResultFlagBits {
    VK_QUERY_RESULT_64_BIT = 0x00000001,
    VK_QUERY_RESULT_WAIT_BIT = 0x00000002,
    VK_QUERY_RESULT_WITH_AVAILABILITY_BIT = 0x00000004,
    VK_QUERY_RESULT_PARTIAL_BIT = 0x00000008,
} VkQueryResultFlagBits;
```

- `VK_QUERY_RESULT_64_BIT` specifies the results will be written as an array of 64-bit unsigned integer values. If this bit is not set, the results will be written as an array of 32-bit unsigned integer values.
- `VK_QUERY_RESULT_WAIT_BIT` specifies that Vulkan will wait for each query's status to become available before retrieving its results.
- `VK_QUERY_RESULT_WITH_AVAILABILITY_BIT` specifies that the availability status accompanies the results.
- `VK_QUERY_RESULT_PARTIAL_BIT` specifies that returning partial results is acceptable.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkQueryResultFlags;
```

`VkQueryResultFlags` is a bitmask type for setting a mask of zero or more `VkQueryResultFlagBits`.

To copy query statuses and numerical results directly to buffer memory, call:

```
// Provided by VK_VERSION_1_0
void vkCmdCopyQueryPoolResults(
    VkCommandBuffer          commandBuffer,
    VkQueryPool              queryPool,
    uint32_t                 firstQuery,
    uint32_t                 queryCount,
    VkBuffer                  dstBuffer,
    VkDeviceSize              dstOffset,
    VkDeviceSize              stride,
    VkQueryResultFlags        flags);
```

- `commandBuffer` is the command buffer into which this command will be recorded.
- `queryPool` is the query pool managing the queries containing the desired results.
- `firstQuery` is the initial query index.
- `queryCount` is the number of queries. `firstQuery` and `queryCount` together define a range of queries.
- `dstBuffer` is a `VkBuffer` object that will receive the results of the copy command.
- `dstOffset` is an offset into `dstBuffer`.
- `stride` is the stride in bytes between results for individual queries within `dstBuffer`. The required size of the backing memory for `dstBuffer` is determined as described above for `vkGetQueryPoolResults`.
- `flags` is a bitmask of `VkQueryResultFlagBits` specifying how and when results are returned.

Any results written for a query are written according to [a layout dependent on the query type](#).

Results for any query in `queryPool` identified by `firstQuery` and `queryCount` that is available are copied to `dstBuffer`.

If `VK_QUERY_RESULT_WITH_AVAILABILITY_BIT` is set, results for all queries in `queryPool` identified by `firstQuery` and `queryCount` are copied to `dstBuffer`, along with an extra availability value written directly after the results of each query and interpreted as an unsigned integer. A value of zero indicates that the results are not yet available, otherwise the query is complete and results are available.

Results for any available query written by this command are final and represent the final result of the query. If `VK_QUERY_RESULT_PARTIAL_BIT` is set, then for any query that is unavailable, an intermediate result between zero and the final result value is written for that query. Otherwise, any result written by this command is undefined.

If `VK_QUERY_RESULT_64_BIT` is set, results and availability values for all queries are written as an array of 64-bit values. If the `queryPool` was created with `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, results for each query are written as an array of the type indicated by `VkPerformanceCounterKHR::storage` for the counter being queried. Otherwise, results and availability values are written as an array of 32-bit values. If an unsigned integer query's value overflows the result type, the value **may** either wrap or saturate. If a signed integer query's value overflows the result type, the value is undefined. If a floating-point query's value is not representable as the result type, the value is undefined.

This command defines an execution dependency between other query commands that reference the same query.

The first [synchronization scope](#) includes all commands which reference the queries in `queryPool` indicated by `query` that occur earlier in [submission order](#). If `flags` does not include `VK_QUERY_RESULT_WAIT_BIT`, `vkCmdEndQuery`, `vkCmdWriteTimestamp2KHR`, and `vkCmdWriteTimestamp` are excluded from this scope.

The second [synchronization scope](#) includes all commands which reference the queries in `queryPool` indicated by `query` that occur later in [submission order](#).

The operation of this command happens after the first scope and happens before the second scope.

`vkCmdCopyQueryPoolResults` is considered to be a transfer operation, and its writes to buffer memory **must** be synchronized using `VK_PIPELINE_STAGE_TRANSFER_BIT` and `VK_ACCESS_TRANSFER_WRITE_BIT` before using the results.

Valid Usage

- VUID-vkCmdCopyQueryPoolResults-firstQuery-09436
`firstQuery` **must** be less than the number of queries in `queryPool`
- VUID-vkCmdCopyQueryPoolResults-firstQuery-09437
The sum of `firstQuery` and `queryCount` **must** be less than or equal to the number of queries in `queryPool`
- VUID-vkCmdCopyQueryPoolResults-queryCount-09438
If `queryCount` is greater than 1, `stride` **must** not be zero
- VUID-vkCmdCopyQueryPoolResults-queryType-09439
If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_TIMESTAMP`, `flags` **must** not contain `VK_QUERY_RESULT_PARTIAL_BIT`
- VUID-vkCmdCopyQueryPoolResults-queryType-09440
If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, `flags` **must** not contain `VK_QUERY_RESULT_WITH_AVAILABILITY_BIT`, `VK_QUERY_RESULT_PARTIAL_BIT`, or `VK_QUERY_RESULT_64_BIT`
- VUID-vkCmdCopyQueryPoolResults-queryType-09441
If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, the `queryPool` **must** have been recorded once for each pass as retrieved via a call to `vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR`

- VUID-vkCmdCopyQueryPoolResults-None-09402
All queries used by the command **must** not be uninitialized when the command is executed
- VUID-vkCmdCopyQueryPoolResults-dstOffset-00819
dstOffset **must** be less than the size of **dstBuffer**
- VUID-vkCmdCopyQueryPoolResults-flags-00822
If **VK_QUERY_RESULT_64_BIT** is not set in **flags** then **dstOffset** and **stride** **must** be multiples of 4
- VUID-vkCmdCopyQueryPoolResults-flags-00823
If **VK_QUERY_RESULT_64_BIT** is set in **flags** then **dstOffset** and **stride** **must** be multiples of 8
- VUID-vkCmdCopyQueryPoolResults-dstBuffer-00824
dstBuffer **must** have enough storage, from **dstOffset**, to contain the result of each query, as described [here](#)
- VUID-vkCmdCopyQueryPoolResults-dstBuffer-00825
dstBuffer **must** have been created with **VK_BUFFER_USAGE_TRANSFER_DST_BIT** usage flag
- VUID-vkCmdCopyQueryPoolResults-dstBuffer-00826
If **dstBuffer** is non-sparse then it **must** be bound completely and contiguously to a single **VkDeviceMemory** object
- VUID-vkCmdCopyQueryPoolResults-queryType-03232
If the **queryType** used to create **queryPool** was **VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR**, **VkPhysicalDevicePerformanceQueryPropertiesKHR::allowCommandBufferQueryCopies** **must** be **VK_TRUE**
- VUID-vkCmdCopyQueryPoolResults-None-07429
All queries used by the command **must** not be active
- VUID-vkCmdCopyQueryPoolResults-None-08752
All queries used by the command **must** have been made *available* by prior executed commands

Valid Usage (Implicit)

- VUID-vkCmdCopyQueryPoolResults-commandBuffer-parameter
commandBuffer **must** be a valid **VkCommandBuffer** handle
- VUID-vkCmdCopyQueryPoolResults-queryPool-parameter
queryPool **must** be a valid **VkQueryPool** handle
- VUID-vkCmdCopyQueryPoolResults-dstBuffer-parameter
dstBuffer **must** be a valid **VkBuffer** handle
- VUID-vkCmdCopyQueryPoolResults-flags-parameter
flags **must** be a valid combination of **VkQueryResultFlagBits** values
- VUID-vkCmdCopyQueryPoolResults-commandBuffer-recording
commandBuffer **must** be in the **recording state**
- VUID-vkCmdCopyQueryPoolResults-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` operations

- `VUID-vkCmdCopyQueryPoolResults-renderpass`
This command **must** only be called outside of a render pass instance
- `VUID-vkCmdCopyQueryPoolResults-commonparent`
Each of `commandBuffer`, `dstBuffer`, and `queryPool` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	<code>VK_QUEUE_COMPUTE_BIT</code> <code>VK_QUEUE_GRAPHICS_BIT</code>	Action

Rendering operations such as clears, MSAA resolves, attachment load/store operations, and blits **may** count towards the results of queries. This behavior is implementation-dependent and **may** vary depending on the path used within an implementation. For example, some implementations have several types of clears, some of which **may** include vertices and some not.

18.3. Occlusion Queries

Occlusion queries track the number of samples that pass the per-fragment tests for a set of drawing commands. As such, occlusion queries are only available on queue families supporting graphics operations. The application **can** then use these results to inform future rendering decisions. An occlusion query is begun and ended by calling `vkCmdBeginQuery` and `vkCmdEndQuery`, respectively. When an occlusion query begins, the count of passing samples always starts at zero. For each drawing command, the count is incremented as described in [Sample Counting](#). If `flags` does not contain `VK_QUERY_CONTROL_PRECISE_BIT` an implementation **may** generate any non-zero result value for the query if the count of passing samples is non-zero.

NOTE

Not setting `VK_QUERY_CONTROL_PRECISE_BIT` mode **may** be more efficient on some implementations, and **should** be used where it is sufficient to know a boolean result on whether any samples passed the per-fragment tests. In this case, some implementations **may** only return zero or one, indifferent to the actual number of

samples passing the per-fragment tests.

Setting `VK_QUERY_CONTROL_PRECISE_BIT` does not guarantee that different implementations return the same number of samples in an occlusion query. Some implementations may kill fragments in the [pre-rasterization shader stage](#), and these killed fragments do not contribute to the final result of the query. It is possible that some implementations generate a zero result value for the query, while others generate a non-zero value.

When an occlusion query finishes, the result for that query is marked as available. The application **can** then either copy the result to a buffer (via `vkCmdCopyQueryPoolResults`) or request it be put into host memory (via `vkGetQueryPoolResults`).

NOTE

If occluding geometry is not drawn first, samples **can** pass the depth test, but still not be visible in a final image.

18.4. Pipeline Statistics Queries

Pipeline statistics queries allow the application to sample a specified set of `VkPipeline` counters. These counters are accumulated by Vulkan for a set of either drawing or dispatching commands while a pipeline statistics query is active. As such, pipeline statistics queries are available on queue families supporting either graphics or compute operations. The availability of pipeline statistics queries is indicated by the `pipelineStatisticsQuery` member of the `VkPhysicalDeviceFeatures` object (see `vkGetPhysicalDeviceFeatures` and `vkCreateDevice` for detecting and requesting this query type on a `VkDevice`).

A pipeline statistics query is begun and ended by calling `vkCmdBeginQuery` and `vkCmdEndQuery`, respectively. When a pipeline statistics query begins, all statistics counters are set to zero. While the query is active, the pipeline type determines which set of statistics are available, but these **must** be configured on the query pool when it is created. If a statistic counter is issued on a command buffer that does not support the corresponding operation, or the counter corresponds to a shading stage which is missing from any of the pipelines used while the query is active, the value of that counter is undefined after the query has been made available. At least one statistic counter relevant to the operations supported on the recording command buffer **must** be enabled.

Bits which **can** be set in `VkQueryPoolCreateInfo::pipelineStatistics` for query pools and in `VkCommandBufferInheritanceInfo::pipelineStatistics` for secondary command buffers, individually enabling pipeline statistics counters, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkQueryPipelineStatisticFlagBits {
    VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT = 0x00000001,
    VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT = 0x00000002,
    VK_QUERY_PIPELINE_STATISTIC_VERTEX_SHADER_INVOCATIONS_BIT = 0x00000004,
    VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT = 0x00000008,
    VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_PRIMITIVES_BIT = 0x00000010,
    VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT = 0x00000020,
    VK_QUERY_PIPELINE_STATISTIC_CLIPPING_PRIMITIVES_BIT = 0x00000040,
    VK_QUERY_PIPELINE_STATISTIC_FRAGMENT_SHADER_INVOCATIONS_BIT = 0x00000080,
    VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_CONTROL_SHADER_PATCHES_BIT = 0x00000100,
    VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_EVALUATION_SHADER_INVOCATIONS_BIT =
0x00000200,
    VK_QUERY_PIPELINE_STATISTIC_COMPUTE_SHADER_INVOCATIONS_BIT = 0x00000400,
} VkQueryPipelineStatisticFlagBits;
```

- **VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT** specifies that queries managed by the pool will count the number of vertices processed by the **input assembly** stage. Vertices corresponding to incomplete primitives **may** contribute to the count.
- **VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT** specifies that queries managed by the pool will count the number of primitives processed by the **input assembly** stage. If primitive restart is enabled, restarting the primitive topology has no effect on the count. Incomplete primitives **may** be counted.
- **VK_QUERY_PIPELINE_STATISTIC_VERTEX_SHADER_INVOCATIONS_BIT** specifies that queries managed by the pool will count the number of vertex shader invocations. This counter's value is incremented each time a vertex shader is **invoked**.
- **VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT** specifies that queries managed by the pool will count the number of geometry shader invocations. This counter's value is incremented each time a geometry shader is **invoked**. In the case of **instanced geometry shaders**, the geometry shader invocations count is incremented for each separate instanced invocation.
- **VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_PRIMITIVES_BIT** specifies that queries managed by the pool will count the number of primitives generated by geometry shader invocations. The counter's value is incremented each time the geometry shader emits a primitive. Restarting primitive topology using the SPIR-V instructions **OpEndPrimitive** or **OpEndStreamPrimitive** has no effect on the geometry shader output primitives count.
- **VK_QUERY_PIPELINE_STATISTIC_CLIPPING_INVOCATIONS_BIT** specifies that queries managed by the pool will count the number of primitives processed by the **Primitive Clipping** stage of the pipeline. The counter's value is incremented each time a primitive reaches the primitive clipping stage.
- **VK_QUERY_PIPELINE_STATISTIC_CLIPPING_PRIMITIVES_BIT** specifies that queries managed by the pool will count the number of primitives output by the **Primitive Clipping** stage of the pipeline. The counter's value is incremented each time a primitive passes the primitive clipping stage. The actual number of primitives output by the primitive clipping stage for a particular input primitive is implementation-dependent but **must** satisfy the following conditions:

- If at least one vertex of the input primitive lies inside the clipping volume, the counter is incremented by one or more.
- Otherwise, the counter is incremented by zero or more.
- `VK_QUERY_PIPELINE_STATISTIC_FRAGMENT_SHADER_INVOCATIONS_BIT` specifies that queries managed by the pool will count the number of fragment shader invocations. The counter's value is incremented each time the fragment shader is [invoked](#).
- `VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_CONTROL_SHADER_PATCHES_BIT` specifies that queries managed by the pool will count the number of patches processed by the tessellation control shader. The counter's value is incremented once for each patch for which a tessellation control shader is [invoked](#).
- `VK_QUERY_PIPELINE_STATISTIC_TESSELLATION_EVALUATION_SHADER_INVOCATIONS_BIT` specifies that queries managed by the pool will count the number of invocations of the tessellation evaluation shader. The counter's value is incremented each time the tessellation evaluation shader is [invoked](#).
- `VK_QUERY_PIPELINE_STATISTIC_COMPUTE_SHADER_INVOCATIONS_BIT` specifies that queries managed by the pool will count the number of compute shader invocations. The counter's value is incremented every time the compute shader is invoked. Implementations **may** skip the execution of certain compute shader invocations or execute additional compute shader invocations for implementation-dependent reasons as long as the results of rendering otherwise remain unchanged.

These values are intended to measure relative statistics on one implementation. Various device architectures will count these values differently. Any or all counters **may** be affected by the issues described in [Query Operation](#).

NOTE

For example, tile-based rendering devices **may** need to replay the scene multiple times, affecting some of the counts.

If a pipeline has `rasterizerDiscardEnable` enabled, implementations **may** discard primitives after the final [pre-rasterization shader stage](#). As a result, if `rasterizerDiscardEnable` is enabled, the clipping input and output primitives counters **may** not be incremented.

When a pipeline statistics query finishes, the result for that query is marked as available. The application **can** copy the result to a buffer (via `vkCmdCopyQueryPoolResults`), or request it be put into host memory (via `vkGetQueryPoolResults`).

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkQueryPipelineStatisticFlags;
```

`VkQueryPipelineStatisticFlags` is a bitmask type for setting a mask of zero or more [VkQueryPipelineStatisticFlagBits](#).

18.5. Timestamp Queries

Timestamps provide applications with a mechanism for monotonically tracking the execution of

commands. A timestamp is an integer value generated by the `VkPhysicalDevice`. Unlike other queries, timestamps do not operate over a range, and so do not use `vkCmdBeginQuery` or `vkCmdEndQuery`. The mechanism is built around a set of commands that allow the application to tell the `VkPhysicalDevice` to write timestamp values to a `query pool` and then either read timestamp values on the host (using `vkGetQueryPoolResults`) or copy timestamp values to a `VkBuffer` (using `vkCmdCopyQueryPoolResults`).

The number of valid bits in a timestamp value is determined by the `VkQueueFamilyProperties::timestampValidBits` property of the queue on which the timestamp is written. Timestamps are supported on any queue which reports a non-zero value for `timestampValidBits` via `vkGetPhysicalDeviceQueueFamilyProperties`. If the `timestampComputeAndGraphics` limit is `VK_TRUE`, timestamps are supported by every queue family that supports either graphics or compute operations (see `VkQueueFamilyProperties`).

The number of nanoseconds it takes for a timestamp value to be incremented by 1 **can** be obtained from `VkPhysicalDeviceLimits::timestampPeriod` after a call to `vkGetPhysicalDeviceProperties`.

To request a timestamp and write the value to memory, call:

```
// Provided by VK_KHR_synchronization2
void vkCmdWriteTimestamp2KHR(
    VkCommandBuffer          commandBuffer,
    VkPipelineStageFlags2    stage,
    VkQueryPool              queryPool,
    uint32_t                 query);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `stage` specifies a stage of the pipeline.
- `queryPool` is the query pool that will manage the timestamp.
- `query` is the query within the query pool that will contain the timestamp.

When `vkCmdWriteTimestamp2KHR` is submitted to a queue, it defines an execution dependency on commands that were submitted before it, and writes a timestamp to a query pool.

The first `synchronization scope` includes all commands that occur earlier in `submission order`. The synchronization scope is limited to operations on the pipeline stage specified by `stage`.

The second `synchronization scope` includes only the timestamp write operation.

NOTE

Implementations may write the timestamp at any stage that is `logically later` than `stage`.

Any timestamp write that `happens-after` another timestamp write in the same submission **must** not have a lower value unless its value overflows the maximum supported integer bit width of the query. If `VK_KHR_calibrated_timestamps` is enabled, this extends to timestamp writes across all submissions on the same logical device: any timestamp write that `happens-after` another **must** not have a lower value unless its value overflows the maximum supported integer bit width of the

query. Timestamps written by this command **must** be in the `VK_TIME_DOMAIN_DEVICE_KHR` time domain. If an overflow occurs, the timestamp value **must** wrap back to zero.

If `vkCmdWriteTimestamp2KHR` is called while executing a render pass instance that has multiview enabled, the timestamp uses N consecutive query indices in the query pool (starting at `query`) where N is the number of bits set in the view mask of the subpass the command is executed in. The resulting query values are determined by an implementation-dependent choice of one of the following behaviors:

- The first query is a timestamp value and (if more than one bit is set in the view mask) zero is written to the remaining queries.
- All N queries are timestamp values.

Either way, if two timestamps are written in the same subpass with multiview enabled, each of the N consecutive queries written for a timestamp **must** not have a lower value than the queries with corresponding indices written by the timestamp that happens-before unless the value overflows the maximum supported integer bit width of the query.

Valid Usage

- VUID-vkCmdWriteTimestamp2-stage-03929
If the `geometryShader` feature is not enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT`
- VUID-vkCmdWriteTimestamp2-stage-03930
If the `tessellationShader` feature is not enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdWriteTimestamp2-stage-07317
If the `attachmentFragmentShadingRate` feature is not enabled, `stage` **must** not contain `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdWriteTimestamp2-synchronization2-03858
The `synchronization2` feature **must** be enabled
- VUID-vkCmdWriteTimestamp2-stage-03859
`stage` **must** only include a single pipeline stage
- VUID-vkCmdWriteTimestamp2-stage-03860
`stage` **must** only include stages valid for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdWriteTimestamp2-queryPool-03861
`queryPool` **must** have been created with a `queryType` of `VK_QUERY_TYPE_TIMESTAMP`
- VUID-vkCmdWriteTimestamp2-timestampValidBits-03863
The command pool's queue family **must** support a non-zero `timestampValidBits`
- VUID-vkCmdWriteTimestamp2-query-04903
`query` **must** be less than the number of queries in `queryPool`
- VUID-vkCmdWriteTimestamp2-None-03864

All queries used by the command **must** be *unavailable*

- VUID-vkCmdWriteTimestamp2-query-03865

If `vkCmdWriteTimestamp2KHR` is called within a render pass instance, the sum of `query` and the number of bits set in the current subpass's view mask **must** be less than or equal to the number of queries in `queryPool`

Valid Usage (Implicit)

- VUID-vkCmdWriteTimestamp2-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdWriteTimestamp2-stage-parameter
`stage` **must** be a valid combination of `VkPipelineStageFlagBits2` values
- VUID-vkCmdWriteTimestamp2-queryPool-parameter
`queryPool` **must** be a valid `VkQueryPool` handle
- VUID-vkCmdWriteTimestamp2-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdWriteTimestamp2-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations
- VUID-vkCmdWriteTimestamp2-commonparent
Both of `commandBuffer`, and `queryPool` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

To request a timestamp and write the value to memory, call:

```
// Provided by VK_VERSION_1_0
void vkCmdWriteTimestamp(
    VkCommandBuffer          commandBuffer,
    VkPipelineStageFlagBits  pipelineStage,
    VkQueryPool              queryPool,
    uint32_t                 query);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pipelineStage` is a `VkPipelineStageFlagBits` value, specifying a stage of the pipeline.
- `queryPool` is the query pool that will manage the timestamp.
- `query` is the query within the query pool that will contain the timestamp.

When `vkCmdWriteTimestamp` is submitted to a queue, it defines an execution dependency on commands that were submitted before it, and writes a timestamp to a query pool.

The first [synchronization scope](#) includes all commands that occur earlier in [submission order](#). The synchronization scope is limited to operations on the pipeline stage specified by `pipelineStage`.

The second [synchronization scope](#) includes only the timestamp write operation.

NOTE

Implementations may write the timestamp at any stage that is [logically later](#) than [stage](#).

Any timestamp write that [happens-after](#) another timestamp write in the same submission **must** not have a lower value unless its value overflows the maximum supported integer bit width of the query. If `VK_KHR_calibrated_timestamps` is enabled, this extends to timestamp writes across all submissions on the same logical device: any timestamp write that [happens-after](#) another **must** not have a lower value unless its value overflows the maximum supported integer bit width of the query. Timestamps written by this command **must** be in the `VK_TIME_DOMAIN_DEVICE_KHR` [time domain](#). If an overflow occurs, the timestamp value **must** wrap back to zero.

If `vkCmdWriteTimestamp` is called while executing a render pass instance that has multiview enabled, the timestamp uses N consecutive query indices in the query pool (starting at `query`) where N is the number of bits set in the view mask of the subpass the command is executed in. The resulting query values are determined by an implementation-dependent choice of one of the following behaviors:

- The first query is a timestamp value and (if more than one bit is set in the view mask) zero is written to the remaining queries.
- All N queries are timestamp values.

Either way, if two timestamps are written in the same subpass with multiview enabled, each of the N consecutive queries written for a timestamp **must** not have a lower value than the queries with corresponding indices written by the timestamp that happens-before unless the value overflows the maximum supported integer bit width of the query.

Valid Usage

- VUID-vkCmdWriteTimestamp-pipelineStage-04074
`pipelineStage` **must** be a `valid stage` for the queue family that was used to create the command pool that `commandBuffer` was allocated from
- VUID-vkCmdWriteTimestamp-pipelineStage-04075
If the `geometryShader` feature is not enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- VUID-vkCmdWriteTimestamp-pipelineStage-04076
If the `tessellationShader` feature is not enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- VUID-vkCmdWriteTimestamp-fragmentShadingRate-07315
If the `attachmentFragmentShadingRate` feature is not enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`
- VUID-vkCmdWriteTimestamp-synchronization2-06489
If the `synchronization2` feature is not enabled, `pipelineStage` **must** not be `VK_PIPELINE_STAGE_NONE`
- VUID-vkCmdWriteTimestamp-queryPool-01416
`queryPool` **must** have been created with a `queryType` of `VK_QUERY_TYPE_TIMESTAMP`
- VUID-vkCmdWriteTimestamp-timestampValidBits-00829
The command pool's queue family **must** support a non-zero `timestampValidBits`
- VUID-vkCmdWriteTimestamp-query-04904
`query` **must** be less than the number of queries in `queryPool`
- VUID-vkCmdWriteTimestamp-None-00830
All queries used by the command **must** be *unavailable*
- VUID-vkCmdWriteTimestamp-query-00831
If `vkCmdWriteTimestamp` is called within a render pass instance, the sum of `query` and the number of bits set in the current subpass's view mask **must** be less than or equal to the number of queries in `queryPool`

Valid Usage (Implicit)

- VUID-vkCmdWriteTimestamp-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdWriteTimestamp-pipelineStage-parameter
`pipelineStage` **must** be a valid `VkPipelineStageFlagBits` value
- VUID-vkCmdWriteTimestamp-queryPool-parameter
`queryPool` **must** be a valid `VkQueryPool` handle
- VUID-vkCmdWriteTimestamp-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`

- VUID-vkCmdWriteTimestamp-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations

- VUID-vkCmdWriteTimestamp-commonparent

Both of `commandBuffer`, and `queryPool` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

18.6. Performance Queries

Performance queries provide applications with a mechanism for getting performance counter information about the execution of command buffers, render passes, and commands.

Each queue family advertises the performance counters that **can** be queried on a queue of that family via a call to `vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR`. Implementations **may** limit access to performance counters based on platform requirements or only to specialized drivers for development purposes.

NOTE

This may include no performance counters being enumerated, or a reduced set. Please refer to platform-specific documentation for guidance on any such restrictions.

Performance queries use the existing `vkCmdBeginQuery` and `vkCmdEndQuery` to control what command buffers, render passes, or commands to get performance information for.

Implementations **may** require multiple passes where the command buffer, render passes, or commands being recorded are the same and are executed on the same queue to record

performance counter data. This is achieved by submitting the same batch and providing a [VkPerformanceQuerySubmitInfoKHR](#) structure containing a counter pass index. The number of passes required for a given performance query pool **can** be queried via a call to [vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR](#).

NOTE

Command buffers recorded with `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT` **must** not be re-submitted. Changing command buffer usage bits **may** affect performance. To avoid this, the application **should** re-record any command buffers with the `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT` when multiple counter passes are required.

Performance counter results from a performance query pool **can** be obtained with the command [vkGetQueryPoolResults](#).

The [VkPerformanceCounterResultKHR](#) union is defined as:

```
// Provided by VK_KHR_performance_query
typedef union VkPerformanceCounterResultKHR {
    int32_t      int32;
    int64_t      int64;
    uint32_t     uint32;
    uint64_t     uint64;
    float        float32;
    double       float64;
} VkPerformanceCounterResultKHR;
```

- `int32` is a 32-bit signed integer value.
- `int64` is a 64-bit signed integer value.
- `uint32` is a 32-bit unsigned integer value.
- `uint64` is a 64-bit unsigned integer value.
- `float32` is a 32-bit floating-point value.
- `float64` is a 64-bit floating-point value.

Performance query results are returned in an array of [VkPerformanceCounterResultKHR](#) unions containing the data associated with each counter in the query, stored in the same order as the counters supplied in `pCounterIndices` when creating the performance query. [VkPerformanceCounterKHR::storage](#) specifies how to parse the counter data.

18.6.1. Profiling Lock

To record and submit a command buffer containing a performance query pool the profiling lock **must** be held. The profiling lock **must** be acquired prior to any call to [vkBeginCommandBuffer](#) that will be using a performance query pool. The profiling lock **must** be held while any command buffer containing a performance query pool is in the *recording*, *executable*, or *pending state*. To acquire the profiling lock, call:

```
// Provided by VK_KHR_performance_query
VkResult vkAcquireProfilingLockKHR(
    VkDevice device,
    const VkAcquireProfilingLockInfoKHR* pInfo);
```

- **device** is the logical device to profile.
- **pInfo** is a pointer to a [VkAcquireProfilingLockInfoKHR](#) structure containing information about how the profiling is to be acquired.

Implementations **may** allow multiple actors to hold the profiling lock concurrently.

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is **VK_TRUE**, **vkAcquireProfilingLockKHR** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage (Implicit)

- VUID-vkAcquireProfilingLockKHR-device-parameter **device** **must** be a valid [VkDevice](#) handle
- VUID-vkAcquireProfilingLockKHR-pInfo-parameter **pInfo** **must** be a valid pointer to a valid [VkAcquireProfilingLockInfoKHR](#) structure

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**
- **VK_TIMEOUT**

The **VkAcquireProfilingLockInfoKHR** structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkAcquireProfilingLockInfoKHR {
    VkStructureType sType;
    const void* pNext;
    VkAcquireProfilingLockFlagsKHR flags;
    uint64_t timeout;
} VkAcquireProfilingLockInfoKHR;
```

- **sType** is a [VkStructureType](#) value identifying this structure.

- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `timeout` indicates how long the function waits, in nanoseconds, if the profiling lock is not available.

Valid Usage (Implicit)

- VUID-VkAcquireProfilingLockInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_ACQUIRE_PROFILING_LOCK_INFO_KHR`
- VUID-VkAcquireProfilingLockInfoKHR-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkAcquireProfilingLockInfoKHR-flags-zero bitmask
`flags` **must** be 0

If `timeout` is 0, `vkAcquireProfilingLockKHR` will not block while attempting to acquire the profiling lock. If `timeout` is `UINT64_MAX`, the function will not return until the profiling lock was acquired.

```
// Provided by VK_KHR_performance_query
typedef enum VkAcquireProfilingLockFlagBitsKHR {
} VkAcquireProfilingLockFlagBitsKHR;
```

```
// Provided by VK_KHR_performance_query
typedef VkFlags VkAcquireProfilingLockFlagsKHR;
```

`VkAcquireProfilingLockFlagsKHR` is a bitmask type for setting a mask, but is currently reserved for future use.

To release the profiling lock, call:

```
// Provided by VK_KHR_performance_query
void vkReleaseProfilingLockKHR(
    VkDevice device);
```

- `device` is the logical device to cease profiling on.

Valid Usage

- VUID-vkReleaseProfilingLockKHR-device-03235
The profiling lock of `device` **must** have been held via a previous successful call to `vkAcquireProfilingLockKHR`

Valid Usage (Implicit)

- VUID-vkReleaseProfilingLockKHR-device-parameter **device** must be a valid [VkDevice](#) handle

Chapter 19. Clear Commands

19.1. Clearing Images Outside a Render Pass Instance

Color and depth/stencil images **can** be cleared outside a render pass instance using [vkCmdClearColorImage](#) or [vkCmdClearDepthStencilImage](#), respectively. These commands are only allowed outside of a render pass instance.

To clear one or more subranges of a color image, call:

```
// Provided by VK_VERSION_1_0
void vkCmdClearColorImage(
    VkCommandBuffer          commandBuffer,
    VkImage                  image,
    VkImageLayout             imageLayout,
    const VkClearColorValue* pColor,
    uint32_t                  rangeCount,
    const VkImageSubresourceRange* pRanges);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `image` is the image to be cleared.
- `imageLayout` specifies the current layout of the image subresource ranges to be cleared, and **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_GENERAL` or `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL`.
- `pColor` is a pointer to a [VkClearColorValue](#) structure containing the values that the image subresource ranges will be cleared to (see [Clear Values](#) below).
- `rangeCount` is the number of image subresource range structures in `pRanges`.
- `pRanges` is a pointer to an array of [VkImageSubresourceRange](#) structures describing a range of mipmap levels, array layers, and aspects to be cleared, as described in [Image Views](#).

Each specified range in `pRanges` is cleared to the value specified by `pColor`.

Valid Usage

- VUID-vkCmdClearColorImage-image-01993
The [format features](#) of `image` **must** contain `VK_FORMAT_FEATURE_TRANSFER_DST_BIT`
- VUID-vkCmdClearColorImage-image-00002
`image` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdClearColorImage-image-01545
`image` **must** not use any of the [formats that require a sampler Y'C_BC_R conversion](#)
- VUID-vkCmdClearColorImage-image-00003
If `image` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdClearColorImage-imageLayout-00004
`imageLayout` **must** specify the layout of the image subresource ranges of `image` specified in `pRanges` at the time this command is executed on a `VkDevice`
- VUID-vkCmdClearColorImage-imageLayout-01394
`imageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-vkCmdClearColorImage-aspectMask-02498
The `VkImageSubresourceRange::aspectMask` members of the elements of the `pRanges` array **must** each only include `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-vkCmdClearColorImage-baseMipLevel-01470
The `VkImageSubresourceRange::baseMipLevel` members of the elements of the `pRanges` array **must** each be less than the `mipLevels` specified in `VkImageCreateInfo` when `image` was created
- VUID-vkCmdClearColorImage-pRanges-01692
For each `VkImageSubresourceRange` element of `pRanges`, if the `levelCount` member is not `VK_REMAINING_MIP_LEVELS`, then `baseMipLevel` + `levelCount` **must** be less than or equal to the `mipLevels` specified in `VkImageCreateInfo` when `image` was created
- VUID-vkCmdClearColorImage-baseArrayLayer-01472
The `VkImageSubresourceRange::baseArrayLayer` members of the elements of the `pRanges` array **must** each be less than the `arrayLayers` specified in `VkImageCreateInfo` when `image` was created
- VUID-vkCmdClearColorImage-pRanges-01693
For each `VkImageSubresourceRange` element of `pRanges`, if the `layerCount` member is not `VK_REMAINING_ARRAY_LAYERS`, then `baseArrayLayer` + `layerCount` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `image` was created
- VUID-vkCmdClearColorImage-image-00007
`image` **must** not have a compressed or depth/stencil format
- VUID-vkCmdClearColorImage-pColor-04961
`pColor` **must** be a valid pointer to a `VkClearColorValue` union
- VUID-vkCmdClearColorImage-commandBuffer-01805
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `image` **must** not be a protected image
- VUID-vkCmdClearColorImage-commandBuffer-01806
If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, **must** not be an unprotected image
- VUID-vkCmdClearColorImage-image-09678
If `image`'s format has components other than R and G, it **must** not have a 64-bit component width

Valid Usage (Implicit)

- VUID-vkCmdClearColorImage-commandBuffer-parameter

`commandBuffer` **must** be a valid `VkCommandBuffer` handle

- VUID-vkCmdClearColorImage-image-parameter
`image` **must** be a valid `VkImage` handle
- VUID-vkCmdClearColorImage-imageLayout-parameter
`imageLayout` **must** be a valid `VkImageLayout` value
- VUID-vkCmdClearColorImage-pRanges-parameter
`pRanges` **must** be a valid pointer to an array of `rangeCount` valid `VkImageSubresourceRange` structures
- VUID-vkCmdClearColorImage-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdClearColorImage-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, or `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdClearColorImage-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdClearColorImage-rangeCount-arraylength
`rangeCount` **must** be greater than 0
- VUID-vkCmdClearColorImage-commonparent
Both of `commandBuffer`, and `image` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT	Action

To clear one or more subranges of a depth/stencil image, call:

```
// Provided by VK_VERSION_1_0
void vkCmdClearDepthStencilImage(
    VkCommandBuffer          commandBuffer,
    VkImage                  image,
    VkImageLayout            imageLayout,
    const VkClearDepthStencilValue* pDepthStencil,
    uint32_t                 rangeCount,
    const VkImageSubresourceRange* pRanges);
```

- **commandBuffer** is the command buffer into which the command will be recorded.
- **image** is the image to be cleared.
- **imageLayout** specifies the current layout of the image subresource ranges to be cleared, and **must** be **VK_IMAGE_LAYOUT_GENERAL** or **VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL**.
- **pDepthStencil** is a pointer to a **VkClearDepthStencilValue** structure containing the values that the depth and stencil image subresource ranges will be cleared to (see **Clear Values** below).
- **rangeCount** is the number of image subresource range structures in **pRanges**.
- **pRanges** is a pointer to an array of **VkImageSubresourceRange** structures describing a range of mipmap levels, array layers, and aspects to be cleared, as described in **Image Views**.

Valid Usage

- VUID-vkCmdClearDepthStencilImage-image-01994
The **format features** of **image** **must** contain **VK_FORMAT_FEATURE_TRANSFER_DST_BIT**
- VUID-vkCmdClearDepthStencilImage-pRanges-02658
If the **aspect** member of any element of **pRanges** includes **VK_IMAGE_ASPECT_STENCIL_BIT**, and **image** was created with **separate stencil usage**, **VK_IMAGE_USAGE_TRANSFER_DST_BIT** **must** have been included in the **VkImageStencilUsageCreateInfo::stencilUsage** used to create **image**
- VUID-vkCmdClearDepthStencilImage-pRanges-02659
If the **aspect** member of any element of **pRanges** includes **VK_IMAGE_ASPECT_STENCIL_BIT**, and **image** was not created with **separate stencil usage**, **VK_IMAGE_USAGE_TRANSFER_DST_BIT** **must** have been included in the **VkImageCreateInfo::usage** used to create **image**
- VUID-vkCmdClearDepthStencilImage-pRanges-02660
If the **aspect** member of any element of **pRanges** includes **VK_IMAGE_ASPECT_DEPTH_BIT**, **VK_IMAGE_USAGE_TRANSFER_DST_BIT** **must** have been included in the **VkImageCreateInfo::usage** used to create **image**
- VUID-vkCmdClearDepthStencilImage-image-00010
If **image** is non-sparse then it **must** be bound completely and contiguously to a single **VkDeviceMemory** object
- VUID-vkCmdClearDepthStencilImage-imageLayout-00011
imageLayout **must** specify the layout of the image subresource ranges of **image** specified in **pRanges** at the time this command is executed on a **VkDevice**
- VUID-vkCmdClearDepthStencilImage-imageLayout-00012

`imageLayout` **must** be either of `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`

- VUID-vkCmdClearDepthStencilImage-aspectMask-02824

The `VkImageSubresourceRange::aspectMask` member of each element of the `pRanges` array **must** not include bits other than `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`

- VUID-vkCmdClearDepthStencilImage-image-02825

If the `image`'s format does not have a stencil component, then the `VkImageSubresourceRange::aspectMask` member of each element of the `pRanges` array **must** not include the `VK_IMAGE_ASPECT_STENCIL_BIT` bit

- VUID-vkCmdClearDepthStencilImage-image-02826

If the `image`'s format does not have a depth component, then the `VkImageSubresourceRange::aspectMask` member of each element of the `pRanges` array **must** not include the `VK_IMAGE_ASPECT_DEPTH_BIT` bit

- VUID-vkCmdClearDepthStencilImage-baseMipLevel-01474

The `VkImageSubresourceRange::baseMipLevel` members of the elements of the `pRanges` array **must** each be less than the `mipLevels` specified in `VkImageCreateInfo` when `image` was created

- VUID-vkCmdClearDepthStencilImage-pRanges-01694

For each `VkImageSubresourceRange` element of `pRanges`, if the `levelCount` member is not `VK_REMAINING_MIP_LEVELS`, then `baseMipLevel + levelCount` **must** be less than or equal to the `mipLevels` specified in `VkImageCreateInfo` when `image` was created

- VUID-vkCmdClearDepthStencilImage-baseArrayLayer-01476

The `VkImageSubresourceRange::baseArrayLayer` members of the elements of the `pRanges` array **must** each be less than the `arrayLayers` specified in `VkImageCreateInfo` when `image` was created

- VUID-vkCmdClearDepthStencilImage-pRanges-01695

For each `VkImageSubresourceRange` element of `pRanges`, if the `layerCount` member is not `VK_REMAINING_ARRAY_LAYERS`, then `baseArrayLayer + layerCount` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `image` was created

- VUID-vkCmdClearDepthStencilImage-image-00014

`image` **must** have a depth/stencil format

- VUID-vkCmdClearDepthStencilImage-commandBuffer-01807

If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `image` **must** not be a protected image

- VUID-vkCmdClearDepthStencilImage-commandBuffer-01808

If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `image` **must** not be an unprotected image

Valid Usage (Implicit)

- VUID-vkCmdClearDepthStencilImage-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle

- VUID-vkCmdClearDepthStencilImage-image-parameter
image **must** be a valid [VkImage](#) handle
- VUID-vkCmdClearDepthStencilImage-imageLayout-parameter
imageLayout **must** be a valid [VkImageLayout](#) value
- VUID-vkCmdClearDepthStencilImage-pDepthStencil-parameter
pDepthStencil **must** be a valid pointer to a valid [VkClearDepthStencilValue](#) structure
- VUID-vkCmdClearDepthStencilImage-pRanges-parameter
pRanges **must** be a valid pointer to an array of **rangeCount** valid [VkImageSubresourceRange](#) structures
- VUID-vkCmdClearDepthStencilImage-commandBuffer-recording
commandBuffer **must** be in the [recording state](#)
- VUID-vkCmdClearDepthStencilImage-commandBuffer-cmdpool
The [VkCommandPool](#) that **commandBuffer** was allocated from **must** support [VK_QUEUE_GRAPHICS_BIT](#) operations
- VUID-vkCmdClearDepthStencilImage-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdClearDepthStencilImage-rangeCount-arraylength
rangeCount **must** be greater than 0
- VUID-vkCmdClearDepthStencilImage-commonparent
Both of **commandBuffer**, and **image** **must** have been created, allocated, or retrieved from the same [VkDevice](#)

Host Synchronization

- Host access to **commandBuffer** **must** be externally synchronized
- Host access to the [VkCommandPool](#) that **commandBuffer** was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_GRAPHIC S_BIT	Action

Clears outside render pass instances are treated as transfer operations for the purposes of memory barriers.

19.2. Clearing Images Inside a Render Pass Instance

To clear one or more regions of color and depth/stencil attachments inside a render pass instance, call:

```
// Provided by VK_VERSION_1_0
void vkCmdClearAttachments(
    VkCommandBuffer          commandBuffer,
    uint32_t                 attachmentCount,
    const VkClearAttachment* pAttachments,
    uint32_t                 rectCount,
    const VkClearRect*       pRects);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `attachmentCount` is the number of entries in the `pAttachments` array.
- `pAttachments` is a pointer to an array of `VkClearAttachment` structures defining the attachments to clear and the clear values to use.
- `rectCount` is the number of entries in the `pRects` array.
- `pRects` is a pointer to an array of `VkClearRect` structures defining regions to clear for every attachment in `pAttachments`.

Unlike other [clear commands](#), `vkCmdClearAttachments` is not a transfer command. It performs its operations in [rasterization order](#). For color attachments, the operations are executed as color attachment writes, by the `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT` stage. For depth/stencil attachments, the operations are executed as [depth writes](#) and [stencil writes](#) by the `VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT` and `VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT` stages.

`vkCmdClearAttachments` is not affected by the bound pipeline state.

NOTE	It is generally preferable to clear attachments by using the <code>VK_ATTACHMENT_LOAD_OP_CLEAR</code> load operation at the start of rendering, as it is more efficient on some implementations.

If any attachment's `aspectMask` to be cleared is not backed by an image view, the clear has no effect on that aspect.

If an attachment being cleared refers to an image view created with an `aspectMask` equal to one of `VK_IMAGE_ASPECT_PLANE_0_BIT`, `VK_IMAGE_ASPECT_PLANE_1_BIT` or `VK_IMAGE_ASPECT_PLANE_2_BIT`, it is considered to be `VK_IMAGE_ASPECT_COLOR_BIT` for purposes of this command, and **must** be cleared with the `VK_IMAGE_ASPECT_COLOR_BIT` aspect as specified by [image view creation](#).

Valid Usage

- VUID-vkCmdClearAttachments-aspectMask-07884

If the `aspectMask` member of any element of `pAttachments` contains `VK_IMAGE_ASPECT_DEPTH_BIT`, the current subpass instance's depth-stencil attachment **must**

be either `VK_ATTACHMENT_UNUSED` or the attachment `format` **must** contain a depth component

- VUID-vkCmdClearAttachments-aspectMask-07885

If the `aspectMask` member of any element of `pAttachments` contains `VK_IMAGE_ASPECT_STENCIL_BIT`, the current subpass instance's depth-stencil attachment **must** be either `VK_ATTACHMENT_UNUSED` or the attachment `format` **must** contain a stencil component

- VUID-vkCmdClearAttachments-aspectMask-07271

If the `aspectMask` member of any element of `pAttachments` contains `VK_IMAGE_ASPECT_COLOR_BIT`, the `colorAttachment` **must** be a valid color attachment index in the current render pass instance

- VUID-vkCmdClearAttachments-rect-02682

The `rect` member of each element of `pRects` **must** have an `extent.width` greater than 0

- VUID-vkCmdClearAttachments-rect-02683

The `rect` member of each element of `pRects` **must** have an `extent.height` greater than 0

- VUID-vkCmdClearAttachments-pRects-00016

The rectangular region specified by each element of `pRects` **must** be contained within the render area of the current render pass instance

- VUID-vkCmdClearAttachments-pRects-06937

The layers specified by each element of `pRects` **must** be contained within every attachment that `pAttachments` refers to, i.e. for each element of `pRects`, `VkClearRect::baseArrayLayer` + `VkClearRect::layerCount` **must** be less than or equal to the number of layers rendered to in the current render pass instance

- VUID-vkCmdClearAttachments-layerCount-01934

The `layerCount` member of each element of `pRects` **must** not be 0

- VUID-vkCmdClearAttachments-commandBuffer-02504

If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, each attachment to be cleared **must** not be a protected image

- VUID-vkCmdClearAttachments-commandBuffer-02505

If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, each attachment to be cleared **must** not be an unprotected image

- VUID-vkCmdClearAttachments-baseArrayLayer-00018

If the render pass instance this is recorded in uses multiview, then `baseArrayLayer` **must** be zero and `layerCount` **must** be one

- VUID-vkCmdClearAttachments-None-09679

If the attachment format has components other than R and G, it **must** not have a 64-bit component width

Valid Usage (Implicit)

- VUID-vkCmdClearAttachments-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdClearAttachments-pAttachments-parameter

`pAttachments` **must** be a valid pointer to an array of `attachmentCount` valid `VkClearAttachment` structures

- VUID-vkCmdClearAttachments-pRects-parameter
`pRects` **must** be a valid pointer to an array of `rectCount` `VkClearRect` structures
- VUID-vkCmdClearAttachments-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdClearAttachments-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdClearAttachments-renderpass
This command **must** only be called inside of a render pass instance
- VUID-vkCmdClearAttachments-attachmentCount-arraylength
`attachmentCount` **must** be greater than 0
- VUID-vkCmdClearAttachments-rectCount-arraylength
`rectCount` **must** be greater than 0

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Inside	VK_QUEUE_GRAPHICS_BIT	Action

The `VkClearRect` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkClearRect {
    VkRect2D    rect;
    uint32_t    baseArrayLayer;
    uint32_t    layerCount;
} VkClearRect;
```

- `rect` is the two-dimensional region to be cleared.
- `baseArrayLayer` is the first layer to be cleared.

- `layerCount` is the number of layers to clear.

The layers [`baseArrayLayer`, `baseArrayLayer` + `layerCount`) counting from the base layer of the attachment image view are cleared.

The `VkClearAttachment` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkClearAttachment {
    VkImageAspectFlags    aspectMask;
    uint32_t              colorAttachment;
    VkClearColorValue     clearColor;
} VkClearAttachment;
```

- `aspectMask` is a mask selecting the color, depth and/or stencil aspects of the attachment to be cleared.
- `colorAttachment` is only meaningful if `VK_IMAGE_ASPECT_COLOR_BIT` is set in `aspectMask`, in which case it is an index into the bound color attachments.
- `clearValue` is the color or depth/stencil value to clear the attachment to, as described in [Clear Values](#) below.

Valid Usage

- VUID-VkClearAttachment-aspectMask-00019
If `aspectMask` includes `VK_IMAGE_ASPECT_COLOR_BIT`, it **must** not include `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-VkClearAttachment-aspectMask-00020
`aspectMask` **must** not include `VK_IMAGE_ASPECT_METADATA_BIT`
- VUID-VkClearAttachment-aspectMask-02246
`aspectMask` **must** not include `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT` for any index *i*

Valid Usage (Implicit)

- VUID-VkClearAttachment-aspectMask-parameter
`aspectMask` **must** be a valid combination of `VkImageAspectFlagBits` values
- VUID-VkClearAttachment-aspectMask-requiredbitmask
`aspectMask` **must** not be 0

19.3. Clear Values

The `VkClearColorValue` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef union VkClearColorValue {
    float        float32[4];
    int32_t      int32[4];
    uint32_t     uint32[4];
} VkClearColorValue;
```

- **float32** are the color clear values when the format of the image or attachment is one of the **numeric formats** with a numeric type that is floating-point. Floating-point values are automatically converted to the format of the image, with the clear value being treated as linear if the image is sRGB.
- **int32** are the color clear values when the format of the image or attachment has a numeric type that is signed integer (**SINT**). Signed integer values are converted to the format of the image by casting to the smaller type (with negative 32-bit values mapping to negative values in the smaller type). If the integer clear value is not representable in the target type (e.g. would overflow in conversion to that type), the clear value is undefined.
- **uint32** are the color clear values when the format of the image or attachment has a numeric type that is unsigned integer (**UINT**). Unsigned integer values are converted to the format of the image by casting to the integer type with fewer bits.

The four array elements of the clear color map to R, G, B, and A components of image formats, in order.

If the image has more than one sample, the same value is written to all samples for any pixels being cleared.

If the image or attachment format has a 64-bit component width, the first 2 array elements of each of the arrays above are reinterpreted as a single 64-bit element for the R component. The next 2 array elements are used in the same way for the G component. In other words, the union behaves as if it had the following additional members:

```
double float64[2];
int64_t int64[2];
uint64_t uint64[2];
```

The **VkClearDepthStencilValue** structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkClearDepthStencilValue {
    float        depth;
    uint32_t     stencil;
} VkClearDepthStencilValue;
```

- **depth** is the clear value for the depth aspect of the depth/stencil attachment. It is a floating-point value which is automatically converted to the attachment's format.

- **stencil** is the clear value for the stencil aspect of the depth/stencil attachment. It is a 32-bit integer value which is converted to the attachment's format by taking the appropriate number of LSBs.

Valid Usage

- VUID-VkClearDepthStencilValue-depth-00022
Unless the `VK_EXT_depth_range_unrestricted` extension is enabled **depth** **must** be between **0.0** and **1.0**, inclusive

The `VkClearColorValue` union is defined as:

```
// Provided by VK_VERSION_1_0
typedef union VkClearColorValue {
    VkClearColorValue    color;
    VkClearDepthStencilValue    depthStencil;
} VkClearColorValue;
```

- **color** specifies the color image clear values to use when clearing a color image or attachment.
- **depthStencil** specifies the depth and stencil clear values to use when clearing a depth/stencil image or attachment.

This union is used where part of the API requires either color or depth/stencil clear values, depending on the attachment, and defines the initial clear values in the `VkRenderPassBeginInfo` structure.

19.4. Filling Buffers

To clear buffer data, call:

```
// Provided by VK_VERSION_1_0
void vkCmdFillBuffer(
    VkCommandBuffer    commandBuffer,
    VkBuffer            dstBuffer,
    VkDeviceSize        dstOffset,
    VkDeviceSize        size,
    uint32_t            data);
```

- **commandBuffer** is the command buffer into which the command will be recorded.
- **dstBuffer** is the buffer to be filled.
- **dstOffset** is the byte offset into the buffer at which to start filling, and **must** be a multiple of 4.
- **size** is the number of bytes to fill, and **must** be either a multiple of 4, or `VK_WHOLE_SIZE` to fill the range from **offset** to the end of the buffer. If `VK_WHOLE_SIZE` is used and the remaining size of the buffer is not a multiple of 4, then the nearest smaller multiple is used.

- **data** is the 4-byte word written repeatedly to the buffer to fill **size** bytes of data. The data word is written to memory according to the host endianness.

vkCmdFillBuffer is treated as a “transfer” operation for the purposes of synchronization barriers. The **VK_BUFFER_USAGE_TRANSFER_DST_BIT** **must** be specified in **usage** of **VkBufferCreateInfo** in order for the buffer to be compatible with **vkCmdFillBuffer**.

Valid Usage

- VUID-vkCmdFillBuffer-dstOffset-00024
dstOffset **must** be less than the size of **dstBuffer**
- VUID-vkCmdFillBuffer-dstOffset-00025
dstOffset **must** be a multiple of 4
- VUID-vkCmdFillBuffer-size-00026
If **size** is not equal to **VK_WHOLE_SIZE**, **size** **must** be greater than 0
- VUID-vkCmdFillBuffer-size-00027
If **size** is not equal to **VK_WHOLE_SIZE**, **size** **must** be less than or equal to the size of **dstBuffer** minus **dstOffset**
- VUID-vkCmdFillBuffer-size-00028
If **size** is not equal to **VK_WHOLE_SIZE**, **size** **must** be a multiple of 4
- VUID-vkCmdFillBuffer-dstBuffer-00029
dstBuffer **must** have been created with **VK_BUFFER_USAGE_TRANSFER_DST_BIT** usage flag
- VUID-vkCmdFillBuffer-apiVersion-07894
VkCommandPool that **commandBuffer** was allocated from **must** support graphics or compute operations
- VUID-vkCmdFillBuffer-dstBuffer-00031
If **dstBuffer** is non-sparse then it **must** be bound completely and contiguously to a single **VkDeviceMemory** object
- VUID-vkCmdFillBuffer-commandBuffer-01811
If **commandBuffer** is an unprotected command buffer and **protectedNoFault** is not supported, **dstBuffer** **must** not be a protected buffer
- VUID-vkCmdFillBuffer-commandBuffer-01812
If **commandBuffer** is a protected command buffer and **protectedNoFault** is not supported, **dstBuffer** **must** not be an unprotected buffer

Valid Usage (Implicit)

- VUID-vkCmdFillBuffer-commandBuffer-parameter
commandBuffer **must** be a valid **VkCommandBuffer** handle
- VUID-vkCmdFillBuffer-dstBuffer-parameter
dstBuffer **must** be a valid **VkBuffer** handle
- VUID-vkCmdFillBuffer-commandBuffer-recording

`commandBuffer` **must** be in the `recording` state

- VUID-vkCmdFillBuffer-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations

- VUID-vkCmdFillBuffer-renderpass

This command **must** only be called outside of a render pass instance

- VUID-vkCmdFillBuffer-commonparent

Both of `commandBuffer`, and `dstBuffer` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

19.5. Updating Buffers

To update buffer data inline in a command buffer, call:

```
// Provided by VK_VERSION_1_0
void vkCmdUpdateBuffer(
    VkCommandBuffer          commandBuffer,
    VkBuffer                 dstBuffer,
    VkDeviceSize             dstOffset,
    VkDeviceSize             dataSize,
    const void*              pData);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `dstBuffer` is a handle to the buffer to be updated.

- `dstOffset` is the byte offset into the buffer to start updating, and **must** be a multiple of 4.
- `dataSize` is the number of bytes to update, and **must** be a multiple of 4.
- `pData` is a pointer to the source data for the buffer update, and **must** be at least `dataSize` bytes in size.

`dataSize` **must** be less than or equal to 65536 bytes. For larger updates, applications **can** use buffer to buffer [copies](#).

NOTE

Buffer updates performed with `vkCmdUpdateBuffer` first copy the data into command buffer memory when the command is recorded (which requires additional storage and may incur an additional allocation), and then copy the data from the command buffer into `dstBuffer` when the command is executed on a device.

The additional cost of this functionality compared to [buffer to buffer copies](#) means it should only be used for very small amounts of data, and is why it is limited to at most 65536 bytes. Applications **can** work around this restriction by issuing multiple `vkCmdUpdateBuffer` commands to different ranges of the same buffer, but doing so is not recommended.

The source data is copied from `pData` to the command buffer when the command is called.

`vkCmdUpdateBuffer` is only allowed outside of a render pass. This command is treated as a “transfer” operation for the purposes of synchronization barriers. The `VK_BUFFER_USAGE_TRANSFER_DST_BIT` **must** be specified in `usage` of `VkBufferCreateInfo` in order for the buffer to be compatible with `vkCmdUpdateBuffer`.

Valid Usage

- VUID-vkCmdUpdateBuffer-dstOffset-00032
`dstOffset` **must** be less than the size of `dstBuffer`
- VUID-vkCmdUpdateBuffer-dataSize-00033
`dataSize` **must** be less than or equal to the size of `dstBuffer` minus `dstOffset`
- VUID-vkCmdUpdateBuffer-dstBuffer-00034
`dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdUpdateBuffer-dstBuffer-00035
If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdUpdateBuffer-dstOffset-00036
`dstOffset` **must** be a multiple of 4
- VUID-vkCmdUpdateBuffer-dataSize-00037
`dataSize` **must** be less than or equal to 65536
- VUID-vkCmdUpdateBuffer-dataSize-00038
`dataSize` **must** be a multiple of 4
- VUID-vkCmdUpdateBuffer-commandBuffer-01813
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported,

`dstBuffer` **must** not be a protected buffer

- VUID-vkCmdUpdateBuffer-commandBuffer-01814

If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstBuffer` **must** not be an unprotected buffer

Valid Usage (Implicit)

- VUID-vkCmdUpdateBuffer-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdUpdateBuffer-dstBuffer-parameter
`dstBuffer` **must** be a valid `VkBuffer` handle
- VUID-vkCmdUpdateBuffer-pData-parameter
`pData` **must** be a valid pointer to an array of `dataSize` bytes
- VUID-vkCmdUpdateBuffer-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdUpdateBuffer-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations
- VUID-vkCmdUpdateBuffer-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdUpdateBuffer-dataSize-arraylength
`dataSize` **must** be greater than 0
- VUID-vkCmdUpdateBuffer-commonparent
Both of `commandBuffer`, and `dstBuffer` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

Chapter 20. Copy Commands

An application **can** copy buffer and image data using several methods described in this chapter, depending on the type of data transfer.

All copy commands are treated as “transfer” operations for the purposes of synchronization barriers.

All copy commands that have a source format with an X component in its format description read undefined values from those bits.

All copy commands that have a destination format with an X component in its format description write undefined values to those bits.

20.1. Copying Data Between Buffers

To copy data between buffer objects, call:

```
// Provided by VK_VERSION_1_0
void vkCmdCopyBuffer(
    VkCommandBuffer          commandBuffer,
    VkBuffer                 srcBuffer,
    VkBuffer                 dstBuffer,
    uint32_t                 regionCount,
    const VkBufferCopy*      pRegions);
```

- **commandBuffer** is the command buffer into which the command will be recorded.
- **srcBuffer** is the source buffer.
- **dstBuffer** is the destination buffer.
- **regionCount** is the number of regions to copy.
- **pRegions** is a pointer to an array of **VkBufferCopy** structures specifying the regions to copy.

Each source region specified by **pRegions** is copied from the source buffer to the destination region of the destination buffer. If any of the specified regions in **srcBuffer** overlaps in memory with any of the specified regions in **dstBuffer**, values read from those overlapping regions are undefined.

Valid Usage

- VUID-vkCmdCopyBuffer-commandBuffer-01822
If **commandBuffer** is an unprotected command buffer and **protectedNoFault** is not supported, **srcBuffer** **must** not be a protected buffer
- VUID-vkCmdCopyBuffer-commandBuffer-01823
If **commandBuffer** is an unprotected command buffer and **protectedNoFault** is not supported, **dstBuffer** **must** not be a protected buffer
- VUID-vkCmdCopyBuffer-commandBuffer-01824

If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstBuffer` **must** not be an unprotected buffer

- VUID-vkCmdCopyBuffer-srcOffset-00113
The `srcOffset` member of each element of `pRegions` **must** be less than the size of `srcBuffer`
- VUID-vkCmdCopyBuffer-dstOffset-00114
The `dstOffset` member of each element of `pRegions` **must** be less than the size of `dstBuffer`
- VUID-vkCmdCopyBuffer-size-00115
The `size` member of each element of `pRegions` **must** be less than or equal to the size of `srcBuffer` minus `srcOffset`
- VUID-vkCmdCopyBuffer-size-00116
The `size` member of each element of `pRegions` **must** be less than or equal to the size of `dstBuffer` minus `dstOffset`
- VUID-vkCmdCopyBuffer-pRegions-00117
The union of the source regions, and the union of the destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-vkCmdCopyBuffer-srcBuffer-00118
`srcBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-vkCmdCopyBuffer-srcBuffer-00119
If `srcBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdCopyBuffer-dstBuffer-00120
`dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdCopyBuffer-dstBuffer-00121
If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

Valid Usage (Implicit)

- VUID-vkCmdCopyBuffer-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdCopyBuffer-srcBuffer-parameter
`srcBuffer` **must** be a valid `VkBuffer` handle
- VUID-vkCmdCopyBuffer-dstBuffer-parameter
`dstBuffer` **must** be a valid `VkBuffer` handle
- VUID-vkCmdCopyBuffer-pRegions-parameter
`pRegions` **must** be a valid pointer to an array of `regionCount` valid `VkBufferCopy` structures
- VUID-vkCmdCopyBuffer-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdCopyBuffer-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations

- VUID-vkCmdCopyBuffer-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyBuffer-regionCount-arraylength
regionCount **must** be greater than 0
- VUID-vkCmdCopyBuffer-commonparent
Each of **commandBuffer**, **dstBuffer**, and **srcBuffer** **must** have been created, allocated, or retrieved from the same **VkDevice**

Host Synchronization

- Host access to **commandBuffer** **must** be externally synchronized
- Host access to the **VkCommandPool** that **commandBuffer** was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

The **VkBufferCopy** structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBufferCopy {
    VkDeviceSize    srcOffset;
    VkDeviceSize    dstOffset;
    VkDeviceSize    size;
} VkBufferCopy;
```

- **srcOffset** is the starting offset in bytes from the start of **srcBuffer**.
- **dstOffset** is the starting offset in bytes from the start of **dstBuffer**.
- **size** is the number of bytes to copy.

Valid Usage

- VUID-VkBufferCopy-size-01988

The **size** **must** be greater than 0

A more extensible version of the copy buffer command is defined below.

To copy data between buffer objects, call:

```
// Provided by VK_KHR_copy_commands2
void vkCmdCopyBuffer2KHR(
    VkCommandBuffer                      commandBuffer,
    const VkCopyBufferInfo2*            pCopyBufferInfo);
```

- **commandBuffer** is the command buffer into which the command will be recorded.
- **pCopyBufferInfo** is a pointer to a [VkCopyBufferInfo2](#) structure describing the copy parameters.

Each source region specified by **pCopyBufferInfo->pRegions** is copied from the source buffer to the destination region of the destination buffer. If any of the specified regions in **pCopyBufferInfo->srcBuffer** overlaps in memory with any of the specified regions in **pCopyBufferInfo->dstBuffer**, values read from those overlapping regions are undefined.

Valid Usage

- VUID-vkCmdCopyBuffer2-commandBuffer-01822
If **commandBuffer** is an unprotected command buffer and **protectedNoFault** is not supported, **srcBuffer** **must** not be a protected buffer
- VUID-vkCmdCopyBuffer2-commandBuffer-01823
If **commandBuffer** is an unprotected command buffer and **protectedNoFault** is not supported, **dstBuffer** **must** not be a protected buffer
- VUID-vkCmdCopyBuffer2-commandBuffer-01824
If **commandBuffer** is a protected command buffer and **protectedNoFault** is not supported, **dstBuffer** **must** not be an unprotected buffer

Valid Usage (Implicit)

- VUID-vkCmdCopyBuffer2-commandBuffer-parameter
commandBuffer **must** be a valid [VkCommandBuffer](#) handle
- VUID-vkCmdCopyBuffer2-pCopyBufferInfo-parameter
pCopyBufferInfo **must** be a valid pointer to a valid [VkCopyBufferInfo2](#) structure
- VUID-vkCmdCopyBuffer2-commandBuffer-recording
commandBuffer **must** be in the [recording state](#)
- VUID-vkCmdCopyBuffer2-commandBuffer-cmdpool
The [VkCommandPool](#) that **commandBuffer** was allocated from **must** support [VK_QUEUE_COMPUTE_BIT](#), [VK_QUEUE_GRAPHICS_BIT](#), or [VK_QUEUE_TRANSFER_BIT](#) operations
- VUID-vkCmdCopyBuffer2-renderpass

This command **must** only be called outside of a render pass instance

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

The `VkCopyBufferInfo2` structure is defined as:

```
typedef struct VkCopyBufferInfo2 {  
    VkStructureType    sType;  
    const void*        pNext;  
    VkBuffer            srcBuffer;  
    VkBuffer            dstBuffer;  
    uint32_t           regionCount;  
    const VkBufferCopy2* pRegions;  
} VkCopyBufferInfo2;
```

```
// Provided by VK_KHR_copy_commands2  
typedef VkCopyBufferInfo2 VkCopyBufferInfo2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcBuffer` is the source buffer.
- `dstBuffer` is the destination buffer.
- `regionCount` is the number of regions to copy.
- `pRegions` is a pointer to an array of `VkBufferCopy2` structures specifying the regions to copy.

Valid Usage

- VUID-VkCopyBufferInfo2-srcOffset-00113
The `srcOffset` member of each element of `pRegions` **must** be less than the size of `srcBuffer`
- VUID-VkCopyBufferInfo2-dstOffset-00114
The `dstOffset` member of each element of `pRegions` **must** be less than the size of `dstBuffer`
- VUID-VkCopyBufferInfo2-size-00115
The `size` member of each element of `pRegions` **must** be less than or equal to the size of `srcBuffer` minus `srcOffset`
- VUID-VkCopyBufferInfo2-size-00116
The `size` member of each element of `pRegions` **must** be less than or equal to the size of `dstBuffer` minus `dstOffset`
- VUID-VkCopyBufferInfo2-pRegions-00117
The union of the source regions, and the union of the destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-VkCopyBufferInfo2-srcBuffer-00118
`srcBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-VkCopyBufferInfo2-srcBuffer-00119
If `srcBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkCopyBufferInfo2-dstBuffer-00120
`dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-VkCopyBufferInfo2-dstBuffer-00121
If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

Valid Usage (Implicit)

- VUID-VkCopyBufferInfo2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_COPY_BUFFER_INFO_2`
- VUID-VkCopyBufferInfo2-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkCopyBufferInfo2-srcBuffer-parameter
`srcBuffer` **must** be a valid `VkBuffer` handle
- VUID-VkCopyBufferInfo2-dstBuffer-parameter
`dstBuffer` **must** be a valid `VkBuffer` handle
- VUID-VkCopyBufferInfo2-pRegions-parameter
`pRegions` **must** be a valid pointer to an array of `regionCount` valid `VkBufferCopy2` structures
- VUID-VkCopyBufferInfo2-regionCount-arraylength
`regionCount` **must** be greater than 0

- VUID-VkCopyBufferInfo2-commonparent

Both of `dstBuffer`, and `srcBuffer` **must** have been created, allocated, or retrieved from the same `VkDevice`

The `VkBufferCopy2` structure is defined as:

```
typedef struct VkBufferCopy2 {  
    VkStructureType    sType;  
    const void*        pNext;  
    VkDeviceSize        srcOffset;  
    VkDeviceSize        dstOffset;  
    VkDeviceSize        size;  
} VkBufferCopy2;
```

```
// Provided by VK_KHR_copy_commands2  
typedef VkBufferCopy2 VkBufferCopy2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcOffset` is the starting offset in bytes from the start of `srcBuffer`.
- `dstOffset` is the starting offset in bytes from the start of `dstBuffer`.
- `size` is the number of bytes to copy.

Valid Usage

- VUID-VkBufferCopy2-size-01988
The `size` **must** be greater than 0

Valid Usage (Implicit)

- VUID-VkBufferCopy2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_COPY_2`
- VUID-VkBufferCopy2-pNext-pNext
`pNext` **must** be `NULL`

20.2. Copying Data Between Images

To copy data between image objects, call:

```
// Provided by VK_VERSION_1_0
void vkCmdCopyImage(
    VkCommandBuffer          commandBuffer,
    VkImage                  srcImage,
    VkImageLayout            srcImageLayout,
    VkImage                  dstImage,
    VkImageLayout            dstImageLayout,
    uint32_t                 regionCount,
    const VkImageCopy*       pRegions);
```

- **commandBuffer** is the command buffer into which the command will be recorded.
- **srcImage** is the source image.
- **srcImageLayout** is the current layout of the source image subresource.
- **dstImage** is the destination image.
- **dstImageLayout** is the current layout of the destination image subresource.
- **regionCount** is the number of regions to copy.
- **pRegions** is a pointer to an array of **VkImageCopy** structures specifying the regions to copy.

Each source region specified by **pRegions** is copied from the source image to the destination region of the destination image. If any of the specified regions in **srcImage** overlaps in memory with any of the specified regions in **dstImage**, values read from those overlapping regions are undefined. If any region accesses a depth aspect in **dstImage** and the **VK_EXT_depth_range_unrestricted** extension is not enabled, values copied from **srcBuffer** outside of the range [0,1] will be written as undefined values to the destination image.

Multi-planar images **can** only be copied on a per-plane basis, and the subresources used in each region when copying to or from such images **must** specify only one plane, though different regions **can** specify different planes. When copying planes of multi-planar images, the format considered is the **compatible format for that plane**, rather than the format of the multi-planar image.

If the format of the destination image has a different **block extent** than the source image (e.g. one is a compressed format), the offset and extent for each of the regions specified is **scaled according to the block extents of each format** to match in size. Copy regions for each image **must** be aligned to a multiple of the texel block extent in each dimension, except at the edges of the image, where region extents **must** match the edge of the image.

Image data **can** be copied between images with different image types. If one image is **VK_IMAGE_TYPE_3D** and the other image is **VK_IMAGE_TYPE_2D** with multiple layers, then each slice is copied to or from a different layer; **depth** slices in the 3D image correspond to **layerCount** layers in the 2D image, with an effective **depth** of **1** used for the 2D image. Other combinations of image types are disallowed.

Valid Usage

- VUID-vkCmdCopyImage-commandBuffer-01825

If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image

- VUID-vkCmdCopyImage-commandBuffer-01826

If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image

- VUID-vkCmdCopyImage-commandBuffer-01827

If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image

- VUID-vkCmdCopyImage-pRegions-00124

The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory

- VUID-vkCmdCopyImage-srcImage-01995

The `format features` of `srcImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_SRC_BIT`

- VUID-vkCmdCopyImage-srcImageLayout-00128

`srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`

- VUID-vkCmdCopyImage-srcImageLayout-01917

`srcImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL`, or `VK_IMAGE_LAYOUT_GENERAL`

- VUID-vkCmdCopyImage-srcImage-09460

If `srcImage` and `dstImage` are the same, and any elements of `pRegions` contains the `srcSubresource` and `dstSubresource` with matching `mipLevel` and overlapping array layers, then the `srcImageLayout` and `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_GENERAL` or `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`

- VUID-vkCmdCopyImage-dstImage-01996

The `format features` of `dstImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_DST_BIT`

- VUID-vkCmdCopyImage-dstImageLayout-00133

`dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`

- VUID-vkCmdCopyImage-dstImageLayout-01395

`dstImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL`, or `VK_IMAGE_LAYOUT_GENERAL`

- VUID-vkCmdCopyImage-srcImage-01548

If the `VkFormat` of each of `srcImage` and `dstImage` is not a `multi-planar format`, the `VkFormat` of each of `srcImage` and `dstImage` **must** be `size-compatible`

- VUID-vkCmdCopyImage-None-01549

In a copy to or from a plane of a `multi-planar image`, the `VkFormat` of the image and plane **must** be compatible according to `the description of compatible planes` for the plane being copied

- VUID-vkCmdCopyImage-srcImage-09247

If the `VkFormat` of each of `srcImage` and `dstImage` is a `compressed image format`, the formats **must** have the same texel block extent

- VUID-vkCmdCopyImage-srcImage-00136
The sample count of `srcImage` and `dstImage` **must** match
- VUID-vkCmdCopyImage-srcOffset-01783
The `srcOffset` and `extent` members of each element of `pRegions` **must** respect the image transfer granularity requirements of `commandBuffer`'s command pool's queue family, as described in [VkQueueFamilyProperties](#)
- VUID-vkCmdCopyImage-dstOffset-01784
The `dstOffset` and `extent` members of each element of `pRegions` **must** respect the image transfer granularity requirements of `commandBuffer`'s command pool's queue family, as described in [VkQueueFamilyProperties](#)
- VUID-vkCmdCopyImage-srcImage-01551
If neither `srcImage` nor `dstImage` has a [multi-planar format](#) then for each element of `pRegions`, `srcSubresource.aspectMask` and `dstSubresource.aspectMask` **must** match
- VUID-vkCmdCopyImage-srcImage-08713
If `srcImage` has a [multi-planar format](#), then for each element of `pRegions`, `srcSubresource.aspectMask` **must** be a single valid [multi-planar aspect mask](#) bit
- VUID-vkCmdCopyImage-dstImage-08714
If `dstImage` has a [multi-planar format](#), then for each element of `pRegions`, `dstSubresource.aspectMask` **must** be a single valid [multi-planar aspect mask](#) bit
- VUID-vkCmdCopyImage-srcImage-01556
If `srcImage` has a [multi-planar format](#) and the `dstImage` does not have a multi-planar image format, then for each element of `pRegions`, `dstSubresource.aspectMask` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-vkCmdCopyImage-dstImage-01557
If `dstImage` has a [multi-planar format](#) and the `srcImage` does not have a multi-planar image format, then for each element of `pRegions`, `srcSubresource.aspectMask` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-vkCmdCopyImage-srcImage-04443
If `srcImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `srcSubresource.baseArrayLayer` **must** be 0 and `srcSubresource.layerCount` **must** be 1
- VUID-vkCmdCopyImage-dstImage-04444
If `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `dstSubresource.baseArrayLayer` **must** be 0 and `dstSubresource.layerCount` **must** be 1
- VUID-vkCmdCopyImage-aspectMask-00142
For each element of `pRegions`, `srcSubresource.aspectMask` **must** specify aspects present in `srcImage`
- VUID-vkCmdCopyImage-aspectMask-00143
For each element of `pRegions`, `dstSubresource.aspectMask` **must** specify aspects present in `dstImage`
- VUID-vkCmdCopyImage-srcOffset-00144
For each element of `pRegions`, `srcOffset.x` and `(extent.width + srcOffset.x)` **must** both be greater than or equal to 0 and less than or equal to the width of the specified `srcSubresource` of `srcImage`

- VUID-vkCmdCopyImage-srcOffset-00145
For each element of `pRegions`, `srcOffset.y` and `(extent.height + srcOffset.y)` **must** both be greater than or equal to `0` and less than or equal to the height of the specified `srcSubresource` of `srcImage`
- VUID-vkCmdCopyImage-srcImage-00146
If `srcImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `srcOffset.y` **must** be `0` and `extent.height` **must** be `1`
- VUID-vkCmdCopyImage-srcOffset-00147
If `srcImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `srcOffset.z` and `(extent.depth + srcOffset.z)` **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `srcSubresource` of `srcImage`
- VUID-vkCmdCopyImage-srcImage-01785
If `srcImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `srcOffset.z` **must** be `0` and `extent.depth` **must** be `1`
- VUID-vkCmdCopyImage-dstImage-01786
If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `dstOffset.z` **must** be `0`
- VUID-vkCmdCopyImage-srcImage-10907
If either the `VkFormat` of each of `srcImage` and `dstImage` is not a `compressed image format`, and `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `extent.depth` **must** be `1`
- VUID-vkCmdCopyImage-srcImage-01787
If `srcImage` is of type `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `srcOffset.z` **must** be `0`
- VUID-vkCmdCopyImage-dstImage-01788
If `dstImage` is of type `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `dstOffset.z` **must** be `0`
- VUID-vkCmdCopyImage-srcImage-07743
If `srcImage` and `dstImage` have a different `VkImageType`, one **must** be `VK_IMAGE_TYPE_3D` and the other **must** be `VK_IMAGE_TYPE_2D`
- VUID-vkCmdCopyImage-srcImage-08793
If `srcImage` and `dstImage` have the same `VkImageType`, for each element of `pRegions`, the `layerCount` members of `srcSubresource` or `dstSubresource` **must** match
- VUID-vkCmdCopyImage-srcImage-01790
If `srcImage` and `dstImage` are both of type `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `extent.depth` **must** be `1`
- VUID-vkCmdCopyImage-srcImage-01791
If `srcImage` is of type `VK_IMAGE_TYPE_2D`, and `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `extent.depth` **must** equal `srcSubresource.layerCount`
- VUID-vkCmdCopyImage-dstImage-01792
If `dstImage` is of type `VK_IMAGE_TYPE_2D`, and `srcImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `extent.depth` **must** equal `dstSubresource.layerCount`
- VUID-vkCmdCopyImage-dstOffset-00150

For each element of `pRegions`, `dstOffset.x` and `(extent.width + dstOffset.x)`, where `extent` is [adjusted for size-compatibility](#), **must** both be greater than or equal to `0` and less than or equal to the width of the specified `dstSubresource` of `dstImage`

- VUID-vkCmdCopyImage-dstOffset-00151

For each element of `pRegions`, `dstOffset.y` and `(extent.height + dstOffset.y)`, where `extent` is [adjusted for size-compatibility](#), **must** both be greater than or equal to `0` and less than or equal to the height of the specified `dstSubresource` of `dstImage`

- VUID-vkCmdCopyImage-dstImage-00152

If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `dstOffset.y` **must** be `0`

- VUID-vkCmdCopyImage-srcImage-10908

If either the `VkFormat` of each of `srcImage` and `dstImage` is not a [compressed image format](#), and `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `extent.height` **must** be `1`, where `extent` is [adjusted for size-compatibility](#)

- VUID-vkCmdCopyImage-dstOffset-00153

If `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `dstOffset.z` and `(extent.depth + dstOffset.z)`, where `extent` is [adjusted for size-compatibility](#), **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `dstSubresource` of `dstImage`

- VUID-vkCmdCopyImage-pRegions-07278

For each element of `pRegions`, `srcOffset.x` **must** be a multiple of the [texel block extent width](#) of the `VkFormat` of `srcImage`

- VUID-vkCmdCopyImage-pRegions-07279

For each element of `pRegions`, `srcOffset.y` **must** be a multiple of the [texel block extent height](#) of the `VkFormat` of `srcImage`

- VUID-vkCmdCopyImage-pRegions-07280

For each element of `pRegions`, `srcOffset.z` **must** be a multiple of the [texel block extent depth](#) of the `VkFormat` of `srcImage`

- VUID-vkCmdCopyImage-pRegions-07281

For each element of `pRegions`, `dstOffset.x` **must** be a multiple of the [texel block extent width](#) of the `VkFormat` of `dstImage`

- VUID-vkCmdCopyImage-pRegions-07282

For each element of `pRegions`, `dstOffset.y` **must** be a multiple of the [texel block extent height](#) of the `VkFormat` of `dstImage`

- VUID-vkCmdCopyImage-pRegions-07283

For each element of `pRegions`, `dstOffset.z` **must** be a multiple of the [texel block extent depth](#) of the `VkFormat` of `dstImage`

- VUID-vkCmdCopyImage-srcImage-01728

For each element of `pRegions`, if the sum of `srcOffset.x` and `extent.width` does not equal the width of the subresource specified by `srcSubresource`, `extent.width` **must** be a multiple of the [texel block extent width](#) of the `VkFormat` of `srcImage`

- VUID-vkCmdCopyImage-srcImage-01729

For each element of `pRegions`, if the sum of `srcOffset.y` and `extent.height` does not equal

the height of the subresource specified by `srcSubresource`, `extent.height` **must** be a multiple of the `texel block extent height` of the `VkFormat` of `srcImage`

- VUID-vkCmdCopyImage-srcImage-01730

For each element of `pRegions`, if the sum of `srcOffset.z` and `extent.depth` does not equal the depth of the subresource specified by `srcSubresource`, `extent.depth` **must** be a multiple of the `texel block extent depth` of the `VkFormat` of `srcImage`

- VUID-vkCmdCopyImage-aspect-06662

If the `aspect` member of any element of `pRegions` includes any flag other than `VK_IMAGE_ASPECT_STENCIL_BIT` or `srcImage` was not created with `separate stencil usage`, `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` **must** have been included in the `VkImageCreateInfo::usage` used to create `srcImage`

- VUID-vkCmdCopyImage-aspect-06663

If the `aspect` member of any element of `pRegions` includes any flag other than `VK_IMAGE_ASPECT_STENCIL_BIT` or `dstImage` was not created with `separate stencil usage`, `VK_IMAGE_USAGE_TRANSFER_DST_BIT` **must** have been included in the `VkImageCreateInfo::usage` used to create `dstImage`

- VUID-vkCmdCopyImage-aspect-06664

If the `aspect` member of any element of `pRegions` includes `VK_IMAGE_ASPECT_STENCIL_BIT`, and `srcImage` was created with `separate stencil usage`, `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` **must** have been included in the `VkImageStencilUsageCreateInfo::stencilUsage` used to create `srcImage`

- VUID-vkCmdCopyImage-aspect-06665

If the `aspect` member of any element of `pRegions` includes `VK_IMAGE_ASPECT_STENCIL_BIT`, and `dstImage` was created with `separate stencil usage`, `VK_IMAGE_USAGE_TRANSFER_DST_BIT` **must** have been included in the `VkImageStencilUsageCreateInfo::stencilUsage` used to create `dstImage`

- VUID-vkCmdCopyImage-srcImage-07966

If `srcImage` is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdCopyImage-srcSubresource-07967

The `srcSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in `VkImageCreateInfo` when `srcImage` was created

- VUID-vkCmdCopyImage-srcSubresource-07968

`srcSubresource.baseArrayLayer` + `srcSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `srcImage` was created

- VUID-vkCmdCopyImage-dstImage-07966

If `dstImage` is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdCopyImage-dstSubresource-07967

The `dstSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in `VkImageCreateInfo` when `dstImage` was created

- VUID-vkCmdCopyImage-dstSubresource-07968

`dstSubresource.baseArrayLayer + dstSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `dstImage` was created

Valid Usage (Implicit)

- VUID-vkCmdCopyImage-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdCopyImage-srcImage-parameter
`srcImage` **must** be a valid `VkImage` handle
- VUID-vkCmdCopyImage-srcImageLayout-parameter
`srcImageLayout` **must** be a valid `VkImageLayout` value
- VUID-vkCmdCopyImage-dstImage-parameter
`dstImage` **must** be a valid `VkImage` handle
- VUID-vkCmdCopyImage-dstImageLayout-parameter
`dstImageLayout` **must** be a valid `VkImageLayout` value
- VUID-vkCmdCopyImage-pRegions-parameter
`pRegions` **must** be a valid pointer to an array of `regionCount` valid `VkImageCopy` structures
- VUID-vkCmdCopyImage-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdCopyImage-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations
- VUID-vkCmdCopyImage-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyImage-regionCount-arraylength
`regionCount` **must** be greater than 0
- VUID-vkCmdCopyImage-commonparent
Each of `commandBuffer`, `dstImage`, and `srcImage` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

The `VkImageCopy` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageCopy {
    VkImageSubresourceLayers srcSubresource;
    VkOffset3D               srcOffset;
    VkImageSubresourceLayers dstSubresource;
    VkOffset3D               dstOffset;
    VkExtent3D               extent;
} VkImageCopy;
```

- `srcSubresource` and `dstSubresource` are `VkImageSubresourceLayers` structures specifying the image subresources of the images used for the source and destination image data, respectively.
- `srcOffset` and `dstOffset` select the initial `x`, `y`, and `z` offsets in texels of the sub-regions of the source and destination image data.
- `extent` is the size in texels of the image to copy in `width`, `height` and `depth`.

Valid Usage

- VUID-VkImageCopy-extent-06668
`extent.width` **must** not be 0
- VUID-VkImageCopy-extent-06669
`extent.height` **must** not be 0
- VUID-VkImageCopy-extent-06670
`extent.depth` **must** not be 0

Valid Usage (Implicit)

- VUID-VkImageCopy-srcSubresource-parameter
`srcSubresource` **must** be a valid `VkImageSubresourceLayers` structure
- VUID-VkImageCopy-dstSubresource-parameter

`dstSubresource` **must** be a valid `VkImageSubresourceLayers` structure

The `VkImageSubresourceLayers` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageSubresourceLayers {
    VkImageAspectFlags    aspectMask;
    uint32_t              mipLevel;
    uint32_t              baseArrayLayer;
    uint32_t              layerCount;
} VkImageSubresourceLayers;
```

- `aspectMask` is a combination of `VkImageAspectFlagBits`, selecting the color, depth and/or stencil aspects to be copied.
- `mipLevel` is the mipmap level to copy
- `baseArrayLayer` and `layerCount` are the starting layer and number of layers to copy.

Valid Usage

- VUID-VkImageSubresourceLayers-aspectMask-00167
If `aspectMask` contains `VK_IMAGE_ASPECT_COLOR_BIT`, it **must** not contain either of `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-VkImageSubresourceLayers-aspectMask-00168
`aspectMask` **must** not contain `VK_IMAGE_ASPECT_METADATA_BIT`
- VUID-VkImageSubresourceLayers-aspectMask-02247
`aspectMask` **must** not include `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT` for any index *i*
- VUID-VkImageSubresourceLayers-layerCount-09243
`layerCount` **must** not be `VK_REMAINING_ARRAY_LAYERS`
- VUID-VkImageSubresourceLayers-layerCount-01700
If `layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, it **must** be greater than 0

Valid Usage (Implicit)

- VUID-VkImageSubresourceLayers-aspectMask-parameter
`aspectMask` **must** be a valid combination of `VkImageAspectFlagBits` values
- VUID-VkImageSubresourceLayers-aspectMask-requiredbitmask
`aspectMask` **must** not be 0

A more extensible version of the copy image command is defined below.

To copy data between image objects, call:

```
// Provided by VK_KHR_copy_commands2
void vkCmdCopyImage2KHR(
    VkCommandBuffer                      commandBuffer,
    const VkCopyImageInfo2*             pCopyImageInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pCopyImageInfo` is a pointer to a [VkCopyImageInfo2](#) structure describing the copy parameters.

This command is functionally identical to [vkCmdCopyImage](#), but includes extensible sub-structures that include `sType` and `pNext` parameters, allowing them to be more easily extended.

Valid Usage

- VUID-vkCmdCopyImage2-commandBuffer-01825
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image
- VUID-vkCmdCopyImage2-commandBuffer-01826
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image
- VUID-vkCmdCopyImage2-commandBuffer-01827
If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image

Valid Usage (Implicit)

- VUID-vkCmdCopyImage2-commandBuffer-parameter
`commandBuffer` **must** be a valid [VkCommandBuffer](#) handle
- VUID-vkCmdCopyImage2-pCopyImageInfo-parameter
`pCopyImageInfo` **must** be a valid pointer to a valid [VkCopyImageInfo2](#) structure
- VUID-vkCmdCopyImage2-commandBuffer-recording
`commandBuffer` **must** be in the [recording state](#)
- VUID-vkCmdCopyImage2-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations
- VUID-vkCmdCopyImage2-renderpass
This command **must** only be called outside of a render pass instance

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally

synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

The `VkCopyImageInfo2` structure is defined as:

```
typedef struct VkCopyImageInfo2 {  
    VkStructureType      sType;  
    const void*          pNext;  
    VkImage               srcImage;  
    VkImageLayout         srcImageLayout;  
    VkImage               dstImage;  
    VkImageLayout         dstImageLayout;  
    uint32_t              regionCount;  
    const VkImageCopy2*   pRegions;  
} VkCopyImageInfo2;
```

```
// Provided by VK_KHR_copy_commands2  
typedef VkCopyImageInfo2 VkCopyImageInfo2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcImage` is the source image.
- `srcImageLayout` is the current layout of the source image subresource.
- `dstImage` is the destination image.
- `dstImageLayout` is the current layout of the destination image subresource.
- `regionCount` is the number of regions to copy.
- `pRegions` is a pointer to an array of `VkImageCopy2` structures specifying the regions to copy.

Valid Usage

- VUID-VkCopyImageInfo2-pRegions-00124

The union of all source regions, and the union of all destination regions, specified by the elements of **pRegions**, **must** not overlap in memory

- VUID-VkCopyImageInfo2-srcImage-01995
The **format features** of **srcImage** **must** contain **VK_FORMAT_FEATURE_TRANSFER_SRC_BIT**
- VUID-VkCopyImageInfo2-srcImageLayout-00128
srcImageLayout **must** specify the layout of the image subresources of **srcImage** specified in **pRegions** at the time this command is executed on a **VkDevice**
- VUID-VkCopyImageInfo2-srcImageLayout-01917
srcImageLayout **must** be **VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR**, **VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL**, or **VK_IMAGE_LAYOUT_GENERAL**
- VUID-VkCopyImageInfo2-srcImage-09460
If **srcImage** and **dstImage** are the same, and any elements of **pRegions** contains the **srcSubresource** and **dstSubresource** with matching **mipLevel** and overlapping array layers, then the **srcImageLayout** and **dstImageLayout** **must** be **VK_IMAGE_LAYOUT_GENERAL** or **VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR**
- VUID-VkCopyImageInfo2-dstImage-01996
The **format features** of **dstImage** **must** contain **VK_FORMAT_FEATURE_TRANSFER_DST_BIT**
- VUID-VkCopyImageInfo2-dstImageLayout-00133
dstImageLayout **must** specify the layout of the image subresources of **dstImage** specified in **pRegions** at the time this command is executed on a **VkDevice**
- VUID-VkCopyImageInfo2-dstImageLayout-01395
dstImageLayout **must** be **VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR**, **VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL**, or **VK_IMAGE_LAYOUT_GENERAL**
- VUID-VkCopyImageInfo2-srcImage-01548
If the **VkFormat** of each of **srcImage** and **dstImage** is not a **multi-planar format**, the **VkFormat** of each of **srcImage** and **dstImage** **must** be **size-compatible**
- VUID-VkCopyImageInfo2-None-01549
In a copy to or from a plane of a **multi-planar image**, the **VkFormat** of the image and plane **must** be compatible according to **the description of compatible planes** for the plane being copied
- VUID-VkCopyImageInfo2-srcImage-09247
If the **VkFormat** of each of **srcImage** and **dstImage** is a **compressed image format**, the formats **must** have the same texel block extent
- VUID-VkCopyImageInfo2-srcImage-00136
The sample count of **srcImage** and **dstImage** **must** match
- VUID-VkCopyImageInfo2-srcOffset-01783
The **srcOffset** and **extent** members of each element of **pRegions** **must** respect the image transfer granularity requirements of **commandBuffer**'s command pool's queue family, as described in **VkQueueFamilyProperties**
- VUID-VkCopyImageInfo2-dstOffset-01784
The **dstOffset** and **extent** members of each element of **pRegions** **must** respect the image transfer granularity requirements of **commandBuffer**'s command pool's queue family, as described in **VkQueueFamilyProperties**

- VUID-VkCopyImageInfo2-srcImage-01551
If neither `srcImage` nor `dstImage` has a [multi-planar format](#) then for each element of `pRegions`, `srcSubresource.aspectMask` and `dstSubresource.aspectMask` **must** match
- VUID-VkCopyImageInfo2-srcImage-08713
If `srcImage` has a [multi-planar format](#), then for each element of `pRegions`, `srcSubresource.aspectMask` **must** be a single valid [multi-planar aspect mask](#) bit
- VUID-VkCopyImageInfo2-dstImage-08714
If `dstImage` has a [multi-planar format](#), then for each element of `pRegions`, `dstSubresource.aspectMask` **must** be a single valid [multi-planar aspect mask](#) bit
- VUID-VkCopyImageInfo2-srcImage-01556
If `srcImage` has a [multi-planar format](#) and the `dstImage` does not have a multi-planar image format, then for each element of `pRegions`, `dstSubresource.aspectMask` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkCopyImageInfo2-dstImage-01557
If `dstImage` has a [multi-planar format](#) and the `srcImage` does not have a multi-planar image format, then for each element of `pRegions`, `srcSubresource.aspectMask` **must** be `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkCopyImageInfo2-srcImage-04443
If `srcImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `srcSubresource.baseArrayLayer` **must** be `0` and `srcSubresource.layerCount` **must** be `1`
- VUID-VkCopyImageInfo2-dstImage-04444
If `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `dstSubresource.baseArrayLayer` **must** be `0` and `dstSubresource.layerCount` **must** be `1`
- VUID-VkCopyImageInfo2-aspectMask-00142
For each element of `pRegions`, `srcSubresource.aspectMask` **must** specify aspects present in `srcImage`
- VUID-VkCopyImageInfo2-aspectMask-00143
For each element of `pRegions`, `dstSubresource.aspectMask` **must** specify aspects present in `dstImage`
- VUID-VkCopyImageInfo2-srcOffset-00144
For each element of `pRegions`, `srcOffset.x` and `(extent.width + srcOffset.x)` **must** both be greater than or equal to `0` and less than or equal to the width of the specified `srcSubresource` of `srcImage`
- VUID-VkCopyImageInfo2-srcOffset-00145
For each element of `pRegions`, `srcOffset.y` and `(extent.height + srcOffset.y)` **must** both be greater than or equal to `0` and less than or equal to the height of the specified `srcSubresource` of `srcImage`
- VUID-VkCopyImageInfo2-srcImage-00146
If `srcImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `srcOffset.y` **must** be `0` and `extent.height` **must** be `1`
- VUID-VkCopyImageInfo2-srcOffset-00147
If `srcImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `srcOffset.z` and `(extent.depth + srcOffset.z)` **must** both be greater than or equal to `0` and less than or

equal to the depth of the specified `srcSubresource` of `srcImage`

- VUID-VkCopyImageInfo2-srcImage-01785
If `srcImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `srcOffset.z` **must** be 0 and `extent.depth` **must** be 1
- VUID-VkCopyImageInfo2-dstImage-01786
If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `dstOffset.z` **must** be 0
- VUID-VkCopyImageInfo2-srcImage-10907
If either the `VkFormat` of each of `srcImage` and `dstImage` is not a `compressed image format`, and `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `extent.depth` **must** be 1
- VUID-VkCopyImageInfo2-srcImage-01787
If `srcImage` is of type `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `srcOffset.z` **must** be 0
- VUID-VkCopyImageInfo2-dstImage-01788
If `dstImage` is of type `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `dstOffset.z` **must** be 0
- VUID-VkCopyImageInfo2-srcImage-07743
If `srcImage` and `dstImage` have a different `VkImageType`, one **must** be `VK_IMAGE_TYPE_3D` and the other **must** be `VK_IMAGE_TYPE_2D`
- VUID-VkCopyImageInfo2-srcImage-08793
If `srcImage` and `dstImage` have the same `VkImageType`, for each element of `pRegions`, the `layerCount` members of `srcSubresource` or `dstSubresource` **must** match
- VUID-VkCopyImageInfo2-srcImage-01790
If `srcImage` and `dstImage` are both of type `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `extent.depth` **must** be 1
- VUID-VkCopyImageInfo2-srcImage-01791
If `srcImage` is of type `VK_IMAGE_TYPE_2D`, and `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `extent.depth` **must** equal `srcSubresource.layerCount`
- VUID-VkCopyImageInfo2-dstImage-01792
If `dstImage` is of type `VK_IMAGE_TYPE_2D`, and `srcImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `extent.depth` **must** equal `dstSubresource.layerCount`
- VUID-VkCopyImageInfo2-dstOffset-00150
For each element of `pRegions`, `dstOffset.x` and $(\text{extent.width} + \text{dstOffset.x})$, where `extent` is `adjusted for size-compatibility`, **must** both be greater than or equal to 0 and less than or equal to the width of the specified `dstSubresource` of `dstImage`
- VUID-VkCopyImageInfo2-dstOffset-00151
For each element of `pRegions`, `dstOffset.y` and $(\text{extent.height} + \text{dstOffset.y})$, where `extent` is `adjusted for size-compatibility`, **must** both be greater than or equal to 0 and less than or equal to the height of the specified `dstSubresource` of `dstImage`
- VUID-VkCopyImageInfo2-dstImage-00152
If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `dstOffset.y` **must** be 0

- VUID-VkCopyImageInfo2-srcImage-10908
If either the `VkFormat` of each of `srcImage` and `dstImage` is not a `compressed image format`, and `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `extent.height` **must** be 1, where `extent` is `adjusted for size-compatibility`
- VUID-VkCopyImageInfo2-dstOffset-00153
If `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `dstOffset.z` and `(extent.depth + dstOffset.z)`, where `extent` is `adjusted for size-compatibility`, **must** both be greater than or equal to 0 and less than or equal to the depth of the specified `dstSubresource` of `dstImage`
- VUID-VkCopyImageInfo2-pRegions-07278
For each element of `pRegions`, `srcOffset.x` **must** be a multiple of the `texel block extent width` of the `VkFormat` of `srcImage`
- VUID-VkCopyImageInfo2-pRegions-07279
For each element of `pRegions`, `srcOffset.y` **must** be a multiple of the `texel block extent height` of the `VkFormat` of `srcImage`
- VUID-VkCopyImageInfo2-pRegions-07280
For each element of `pRegions`, `srcOffset.z` **must** be a multiple of the `texel block extent depth` of the `VkFormat` of `srcImage`
- VUID-VkCopyImageInfo2-pRegions-07281
For each element of `pRegions`, `dstOffset.x` **must** be a multiple of the `texel block extent width` of the `VkFormat` of `dstImage`
- VUID-VkCopyImageInfo2-pRegions-07282
For each element of `pRegions`, `dstOffset.y` **must** be a multiple of the `texel block extent height` of the `VkFormat` of `dstImage`
- VUID-VkCopyImageInfo2-pRegions-07283
For each element of `pRegions`, `dstOffset.z` **must** be a multiple of the `texel block extent depth` of the `VkFormat` of `dstImage`
- VUID-VkCopyImageInfo2-srcImage-01728
For each element of `pRegions`, if the sum of `srcOffset.x` and `extent.width` does not equal the width of the subresource specified by `srcSubresource`, `extent.width` **must** be a multiple of the `texel block extent width` of the `VkFormat` of `srcImage`
- VUID-VkCopyImageInfo2-srcImage-01729
For each element of `pRegions`, if the sum of `srcOffset.y` and `extent.height` does not equal the height of the subresource specified by `srcSubresource`, `extent.height` **must** be a multiple of the `texel block extent height` of the `VkFormat` of `srcImage`
- VUID-VkCopyImageInfo2-srcImage-01730
For each element of `pRegions`, if the sum of `srcOffset.z` and `extent.depth` does not equal the depth of the subresource specified by `srcSubresource`, `extent.depth` **must** be a multiple of the `texel block extent depth` of the `VkFormat` of `srcImage`
- VUID-VkCopyImageInfo2-aspect-06662
If the `aspect` member of any element of `pRegions` includes any flag other than `VK_IMAGE_ASPECT_STENCIL_BIT` or `srcImage` was not created with `separate stencil usage`, `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` **must** have been included in the `VkImageCreateInfo`

::usage used to create srcImage

- VUID-VkCopyImageInfo2-aspect-06663

If the aspect member of any element of pRegions includes any flag other than VK_IMAGE_ASPECT_STENCIL_BIT or dstImage was not created with separate stencil usage, VK_IMAGE_USAGE_TRANSFER_DST_BIT **must** have been included in the VkImageCreateInfo::usage used to create dstImage

- VUID-VkCopyImageInfo2-aspect-06664

If the aspect member of any element of pRegions includes VK_IMAGE_ASPECT_STENCIL_BIT, and srcImage was created with separate stencil usage, VK_IMAGE_USAGE_TRANSFER_SRC_BIT **must** have been included in the VkImageStencilUsageCreateInfo::stencilUsage used to create srcImage

- VUID-VkCopyImageInfo2-aspect-06665

If the aspect member of any element of pRegions includes VK_IMAGE_ASPECT_STENCIL_BIT, and dstImage was created with separate stencil usage, VK_IMAGE_USAGE_TRANSFER_DST_BIT **must** have been included in the VkImageStencilUsageCreateInfo::stencilUsage used to create dstImage

- VUID-VkCopyImageInfo2-srcImage-07966

If srcImage is non-sparse then the image or the specified disjoint plane **must** be bound completely and contiguously to a single VkDeviceMemory object

- VUID-VkCopyImageInfo2-srcSubresource-07967

The srcSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when srcImage was created

- VUID-VkCopyImageInfo2-srcSubresource-07968

srcSubresource.baseArrayLayer + srcSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when srcImage was created

- VUID-VkCopyImageInfo2-dstImage-07966

If dstImage is non-sparse then the image or the specified disjoint plane **must** be bound completely and contiguously to a single VkDeviceMemory object

- VUID-VkCopyImageInfo2-dstSubresource-07967

The dstSubresource.mipLevel member of each element of pRegions **must** be less than the mipLevels specified in VkImageCreateInfo when dstImage was created

- VUID-VkCopyImageInfo2-dstSubresource-07968

dstSubresource.baseArrayLayer + dstSubresource.layerCount of each element of pRegions **must** be less than or equal to the arrayLayers specified in VkImageCreateInfo when dstImage was created

Valid Usage (Implicit)

- VUID-VkCopyImageInfo2-sType-sType

sType **must** be VK_STRUCTURE_TYPE_COPY_IMAGE_INFO_2

- VUID-VkCopyImageInfo2-pNext-pNext

pNext must be NULL

- VUID-VkCopyImageInfo2-srcImage-parameter
srcImage must be a valid [VkImage](#) handle
- VUID-VkCopyImageInfo2-srcImageLayout-parameter
srcImageLayout must be a valid [VkImageLayout](#) value
- VUID-VkCopyImageInfo2-dstImage-parameter
dstImage must be a valid [VkImage](#) handle
- VUID-VkCopyImageInfo2-dstImageLayout-parameter
dstImageLayout must be a valid [VkImageLayout](#) value
- VUID-VkCopyImageInfo2-pRegions-parameter
pRegions must be a valid pointer to an array of **regionCount** valid [VkImageCopy2](#) structures
- VUID-VkCopyImageInfo2-regionCount-arraylength
regionCount must be greater than 0
- VUID-VkCopyImageInfo2-commonparent
Both of **dstImage**, and **srcImage** must have been created, allocated, or retrieved from the same [VkDevice](#)

The [VkImageCopy2](#) structure is defined as:

```
typedef struct VkImageCopy2 {  
    VkStructureType    sType;  
    const void*        pNext;  
    VkImageSubresourceLayers srcSubresource;  
    VkOffset3D          srcOffset;  
    VkImageSubresourceLayers dstSubresource;  
    VkOffset3D          dstOffset;  
    VkExtent3D          extent;  
} VkImageCopy2;
```

```
// Provided by VK_KHR_copy_commands2  
typedef VkImageCopy2 VkImageCopy2KHR;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is NULL or a pointer to a structure extending this structure.
- **srcSubresource** and **dstSubresource** are [VkImageSubresourceLayers](#) structures specifying the image subresources of the images used for the source and destination image data, respectively.
- **srcOffset** and **dstOffset** select the initial **x**, **y**, and **z** offsets in texels of the sub-regions of the source and destination image data.
- **extent** is the size in texels of the image to copy in **width**, **height** and **depth**.

Valid Usage

- VUID-VkImageCopy2-extent-06668
`extent.width` **must** not be 0
- VUID-VkImageCopy2-extent-06669
`extent.height` **must** not be 0
- VUID-VkImageCopy2-extent-06670
`extent.depth` **must** not be 0

Valid Usage (Implicit)

- VUID-VkImageCopy2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_COPY_2`
- VUID-VkImageCopy2-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkImageCopy2-srcSubresource-parameter
`srcSubresource` **must** be a valid `VkImageSubresourceLayers` structure
- VUID-VkImageCopy2-dstSubresource-parameter
`dstSubresource` **must** be a valid `VkImageSubresourceLayers` structure

20.3. Copying Data Between Buffers and Images

Data **can** be copied between buffers and images, enabling applications to load and store data between images and application-defined offsets in buffer memory.

When copying between a buffer and an image, texels in the image and bytes in the buffer are accessed as follows.

Texels at each coordinate (x,y,z,layer) in the image subresource are accessed, where:

x is in the range [`imageOffset.x`, `imageOffset.x + imageExtent.width`),

y is in the range [`imageOffset.y`, `imageOffset.y + imageExtent.height`),

z is in the range [`imageOffset.z`, `imageOffset.z + imageExtent.depth`),

layer is in the range [`imageSubresource.baseArrayLayer`, `imageSubresource.baseArrayLayer + imageSubresource.layerCount`)

For each (x,y,z,layer) coordinate in the image, bytes in the buffer are accessed at offsets in the range

[texelOffset, texelOffset + blockSize), where:

$$\text{texelOffset} = \text{bufferOffset} + ((x - \text{imageOffset.x}) / \text{blockWidth} \times \text{blockSize}) + ((y - \text{imageOffset.y}) / \text{blockHeight} \times \text{rowExtent}) + ((z - \text{imageOffset.z}) / \text{blockDepth} \times \text{sliceExtent}) + ((\text{layer} - \text{imageSubresource.baseArrayLayer}) \times \text{layerExtent})$$
$$\text{rowExtent} = \lceil \max(\text{bufferRowLength}, \text{imageExtent.width}) / \text{blockWidth} \rceil \times \text{blockSize}$$
$$\text{sliceExtent} = \lceil \max(\text{bufferImageHeight}, \text{imageExtent.height}) / \text{blockHeight} \rceil \times \text{rowExtent}$$
$$\text{layerExtent} = \lceil \text{imageExtent.depth} / \text{blockDepth} \rceil \times \text{sliceExtent}$$

and where blockSize, blockWidth, blockHeight, and blockDepth are the [texel block size and extents](#) of the image's format.

When copying between a buffer and the depth or stencil aspect of an image, data in the buffer is assumed to be laid out as separate planes rather than interleaved. Addressing calculations are thus performed for a different format than the base image, according to the aspect, as described in the following table:

Table 24. Depth/Stencil Aspect Copy Table

Base Format	Depth Aspect Format	Stencil Aspect Format
VK_FORMAT_D16_UNORM	VK_FORMAT_D16_UNORM	-
VK_FORMAT_X8_D24_UNORM_PACK32	VK_FORMAT_X8_D24_UNORM_PACK32	-
VK_FORMAT_D32_SFLOAT	VK_FORMAT_D32_SFLOAT	-
VK_FORMAT_S8_UINT	-	VK_FORMAT_S8_UINT
VK_FORMAT_D16_UNORM_S8_UINT	VK_FORMAT_D16_UNORM	VK_FORMAT_S8_UINT
VK_FORMAT_D24_UNORM_S8_UINT	VK_FORMAT_X8_D24_UNORM_PACK32	VK_FORMAT_S8_UINT
VK_FORMAT_D32_SFLOAT_S8_UINT	VK_FORMAT_D32_SFLOAT	VK_FORMAT_S8_UINT

When copying between a buffer and any plane of a [multi-planar image](#), addressing calculations are performed using the [compatible format for that plane](#), rather than the format of the multi-planar image.

Each texel block is copied from one resource to the other according to the above addressing equations.

To copy data from a buffer object to an image object, call:

```
// Provided by VK_VERSION_1_0
void vkCmdCopyBufferToImage(
    VkCommandBuffer          commandBuffer,
    VkBuffer                 srcBuffer,
    VkImage                 dstImage,
    VkImageLayout            dstImageLayout,
    uint32_t                 regionCount,
    const VkBufferImageCopy* pRegions);
```

- **commandBuffer** is the command buffer into which the command will be recorded.
- **srcBuffer** is the source buffer.
- **dstImage** is the destination image.
- **dstImageLayout** is the layout of the destination image subresources for the copy.
- **regionCount** is the number of regions to copy.
- **pRegions** is a pointer to an array of **VkBufferImageCopy** structures specifying the regions to copy.

Each source region specified by **pRegions** is copied from the source buffer to the destination region of the destination image according to the [addressing calculations](#) for each resource. If any of the specified regions in **srcBuffer** overlaps in memory with any of the specified regions in **dstImage**, values read from those overlapping regions are undefined. If any region accesses a depth aspect in **dstImage** and the **VK_EXT_depth_range_unrestricted** extension is not enabled, values copied from **srcBuffer** outside of the range [0,1] will be written as undefined values to the destination image.

Copy regions for the image **must** be aligned to a multiple of the texel block extent in each dimension, except at the edges of the image, where region extents **must** match the edge of the image.

Valid Usage

- VUID-vkCmdCopyBufferToImage-dstImage-07966
If **dstImage** is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single **VkDeviceMemory** object
- VUID-vkCmdCopyBufferToImage-imageSubresource-07967
The **imageSubresource.mipLevel** member of each element of **pRegions** **must** be less than the **mipLevels** specified in **VkImageCreateInfo** when **dstImage** was created
- VUID-vkCmdCopyBufferToImage-imageSubresource-07968
imageSubresource.baseArrayLayer + **imageSubresource.layerCount** of each element of **pRegions** **must** be less than or equal to the **arrayLayers** specified in **VkImageCreateInfo** when **dstImage** was created
- VUID-vkCmdCopyBufferToImage-imageSubresource-07971
For each element of **pRegions**, **imageOffset.x** and **(imageExtent.width + imageOffset.x)** **must** both be greater than or equal to 0 and less than or equal to the width of the specified **imageSubresource** of **dstImage**

- VUID-vkCmdCopyBufferToImage-imageSubresource-07972
For each element of `pRegions`, `imageOffset.y` and `(imageExtent.height + imageOffset.y)` **must** both be greater than or equal to 0 and less than or equal to the height of the specified `imageSubresource` of `dstImage`
- VUID-vkCmdCopyBufferToImage-dstImage-07973
`dstImage` **must** have a sample count equal to `VK_SAMPLE_COUNT_1_BIT`
- VUID-vkCmdCopyBufferToImage-commandBuffer-01828
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcBuffer` **must** not be a protected buffer
- VUID-vkCmdCopyBufferToImage-commandBuffer-01829
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image
- VUID-vkCmdCopyBufferToImage-commandBuffer-01830
If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image
- VUID-vkCmdCopyBufferToImage-commandBuffer-07737
If the queue family used to create the `VkCommandPool` which `commandBuffer` was allocated from does not support `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`, the `bufferOffset` member of any element of `pRegions` **must** be a multiple of 4
- VUID-vkCmdCopyBufferToImage-imageOffset-07738
The `imageOffset` and `imageExtent` members of each element of `pRegions` **must** respect the image transfer granularity requirements of `commandBuffer`'s command pool's queue family, as described in `VkQueueFamilyProperties`
- VUID-vkCmdCopyBufferToImage-commandBuffer-07739
If the queue family used to create the `VkCommandPool` which `commandBuffer` was allocated from does not support `VK_QUEUE_GRAPHICS_BIT`, for each element of `pRegions`, the `aspectMask` member of `imageSubresource` **must** not be `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-vkCmdCopyBufferToImage-pRegions-00171
`srcBuffer` **must** be large enough to contain all buffer locations that are accessed according to `Buffer and Image Addressing`, for each element of `pRegions`
- VUID-vkCmdCopyBufferToImage-pRegions-00173
The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-vkCmdCopyBufferToImage-srcBuffer-00174
`srcBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-vkCmdCopyBufferToImage-dstImage-01997
The `format features` of `dstImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_DST_BIT`
- VUID-vkCmdCopyBufferToImage-srcBuffer-00176
If `srcBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdCopyBufferToImage-dstImage-00177
dstImage **must** have been created with VK_IMAGE_USAGE_TRANSFER_DST_BIT usage flag
- VUID-vkCmdCopyBufferToImage-dstImageLayout-00180
dstImageLayout **must** specify the layout of the image subresources of dstImage specified in pRegions at the time this command is executed on a VkDevice
- VUID-vkCmdCopyBufferToImage-dstImageLayout-01396
dstImageLayout **must** be VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, or VK_IMAGE_LAYOUT_GENERAL
- VUID-vkCmdCopyBufferToImage-pRegions-07931
If VK_EXT_depth_range_unrestricted is not enabled, for each element of pRegions whose imageSubresource contains a depth aspect, the data in srcBuffer **must** be in the range [0,1]
- VUID-vkCmdCopyBufferToImage-dstImage-07979
If dstImage is of type VK_IMAGE_TYPE_1D, then for each element of pRegions, imageOffset.y **must** be 0 and imageExtent.height **must** be 1
- VUID-vkCmdCopyBufferToImage-imageOffset-09104
For each element of pRegions, imageOffset.z and (imageExtent.depth + imageOffset.z) **must** both be greater than or equal to 0 and less than or equal to the depth of the specified imageSubresource of dstImage
- VUID-vkCmdCopyBufferToImage-dstImage-07980
If dstImage is of type VK_IMAGE_TYPE_1D or VK_IMAGE_TYPE_2D, then for each element of pRegions, imageOffset.z **must** be 0 and imageExtent.depth **must** be 1
- VUID-vkCmdCopyBufferToImage-dstImage-07274
For each element of pRegions, imageOffset.x **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-dstImage-07275
For each element of pRegions, imageOffset.y **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-dstImage-07276
For each element of pRegions, imageOffset.z **must** be a multiple of the texel block extent depth of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-dstImage-00207
For each element of pRegions, if the sum of imageOffset.x and extent.width does not equal the width of the subresource specified by imageSubresource, extent.width **must** be a multiple of the texel block extent width of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-dstImage-00208
For each element of pRegions, if the sum of imageOffset.y and extent.height does not equal the height of the subresource specified by imageSubresource, extent.height **must** be a multiple of the texel block extent height of the VkFormat of dstImage
- VUID-vkCmdCopyBufferToImage-dstImage-00209
For each element of pRegions, if the sum of imageOffset.z and extent.depth does not equal the depth of the subresource specified by srcSubresource, extent.depth **must** be a multiple of the texel block extent depth of the VkFormat of dstImage

- VUID-vkCmdCopyBufferToImage-imageSubresource-09105
For each element of `pRegions`, `imageSubresource.aspectMask` **must** specify aspects present in `dstImage`
- VUID-vkCmdCopyBufferToImage-dstImage-07981
If `dstImage` has a `multi-planar format`, then for each element of `pRegions`, `imageSubresource.aspectMask` **must** be a single valid `multi-planar aspect mask` bit
- VUID-vkCmdCopyBufferToImage-dstImage-07983
If `dstImage` is of type `VK_IMAGE_TYPE_3D`, for each element of `pRegions`, `imageSubresource.baseArrayLayer` **must** be 0 and `imageSubresource.layerCount` **must** be 1
- VUID-vkCmdCopyBufferToImage-bufferRowLength-09106
For each element of `pRegions`, `bufferRowLength` **must** be a multiple of the `texel block extent width` of the `VkFormat` of `dstImage`
- VUID-vkCmdCopyBufferToImage-bufferImageHeight-09107
For each element of `pRegions`, `bufferImageHeight` **must** be a multiple of the `texel block extent height` of the `VkFormat` of `dstImage`
- VUID-vkCmdCopyBufferToImage-bufferRowLength-09108
For each element of `pRegions`, `bufferRowLength` divided by the `texel block extent width` and then multiplied by the texel block size of `dstImage` **must** be less than or equal to $2^{31}-1$
- VUID-vkCmdCopyBufferToImage-dstImage-07975
If `dstImage` does not have either a depth/stencil format or a `multi-planar format`, then for each element of `pRegions`, `bufferOffset` **must** be a multiple of the `texel block size`
- VUID-vkCmdCopyBufferToImage-dstImage-07976
If `dstImage` has a `multi-planar format`, then for each element of `pRegions`, `bufferOffset` **must** be a multiple of the element size of the compatible format for the format and the `aspectMask` of the `imageSubresource` as defined in `Compatible Formats of Planes of Multi-Planar Formats`
- VUID-vkCmdCopyBufferToImage-dstImage-07978
If `dstImage` has a depth/stencil format, the `bufferOffset` member of any element of `pRegions` **must** be a multiple of 4

Valid Usage (Implicit)

- VUID-vkCmdCopyBufferToImage-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdCopyBufferToImage-srcBuffer-parameter
`srcBuffer` **must** be a valid `VkBuffer` handle
- VUID-vkCmdCopyBufferToImage-dstImage-parameter
`dstImage` **must** be a valid `VkImage` handle
- VUID-vkCmdCopyBufferToImage-dstImageLayout-parameter
`dstImageLayout` **must** be a valid `VkImageLayout` value
- VUID-vkCmdCopyBufferToImage-pRegions-parameter

`pRegions` **must** be a valid pointer to an array of `regionCount` valid `VkBufferImageCopy` structures

- VUID-vkCmdCopyBufferToImage-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdCopyBufferToImage-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations
- VUID-vkCmdCopyBufferToImage-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdCopyBufferToImage-regionCount-arraylength
`regionCount` **must** be greater than 0
- VUID-vkCmdCopyBufferToImage-commonparent
Each of `commandBuffer`, `dstImage`, and `srcBuffer` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

To copy data from an image object to a buffer object, call:


```
// Provided by VK_VERSION_1_0
void vkCmdCopyImageToBuffer(
    VkCommandBuffer          commandBuffer,
    VkImage                  srcImage,
    VkImageLayout            srcImageLayout,
    VkBuffer                 dstBuffer,
    uint32_t                 regionCount,
    const VkBufferImageCopy* pRegions);
```

- **commandBuffer** is the command buffer into which the command will be recorded.
- **srcImage** is the source image.
- **srcImageLayout** is the layout of the source image subresources for the copy.
- **dstBuffer** is the destination buffer.
- **regionCount** is the number of regions to copy.
- **pRegions** is a pointer to an array of **VkBufferImageCopy** structures specifying the regions to copy.

Each source region specified by **pRegions** is copied from the source image to the destination region of the destination buffer according to the [addressing calculations](#) for each resource. If any of the specified regions in **srcImage** overlaps in memory with any of the specified regions in **dstBuffer**, values read from those overlapping regions are undefined.

Copy regions for the image **must** be aligned to a multiple of the texel block extent in each dimension, except at the edges of the image, where region extents **must** match the edge of the image.

Valid Usage

- VUID-vkCmdCopyImageToBuffer-srcImage-07966
If **srcImage** is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single **VkDeviceMemory** object
- VUID-vkCmdCopyImageToBuffer-imageSubresource-07967
The **imageSubresource.mipLevel** member of each element of **pRegions** **must** be less than the **mipLevels** specified in **VkImageCreateInfo** when **srcImage** was created
- VUID-vkCmdCopyImageToBuffer-imageSubresource-07968
imageSubresource.baseArrayLayer + **imageSubresource.layerCount** of each element of **pRegions** **must** be less than or equal to the **arrayLayers** specified in **VkImageCreateInfo** when **srcImage** was created
- VUID-vkCmdCopyImageToBuffer-imageSubresource-07971
For each element of **pRegions**, **imageOffset.x** and **(imageExtent.width + imageOffset.x)** **must** both be greater than or equal to **0** and less than or equal to the width of the specified **imageSubresource** of **srcImage**
- VUID-vkCmdCopyImageToBuffer-imageSubresource-07972
For each element of **pRegions**, **imageOffset.y** and **(imageExtent.height + imageOffset.y)**

must both be greater than or equal to 0 and less than or equal to the height of the specified `imageSubresource` of `srcImage`

- VUID-vkCmdCopyImageToBuffer-srcImage-07973
`srcImage` **must** have a sample count equal to `VK_SAMPLE_COUNT_1_BIT`
- VUID-vkCmdCopyImageToBuffer-commandBuffer-01831
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image
- VUID-vkCmdCopyImageToBuffer-commandBuffer-01832
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstBuffer` **must** not be a protected buffer
- VUID-vkCmdCopyImageToBuffer-commandBuffer-01833
If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstBuffer` **must** not be an unprotected buffer
- VUID-vkCmdCopyImageToBuffer-commandBuffer-07746
If the queue family used to create the `VkCommandPool` which `commandBuffer` was allocated from does not support `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`, the `bufferOffset` member of any element of `pRegions` **must** be a multiple of 4
- VUID-vkCmdCopyImageToBuffer-imageOffset-07747
The `imageOffset` and `imageExtent` members of each element of `pRegions` **must** respect the image transfer granularity requirements of `commandBuffer`'s command pool's queue family, as described in `VkQueueFamilyProperties`
- VUID-vkCmdCopyImageToBuffer-commandBuffer-10216
If the queue family used to create the `VkCommandPool` which `commandBuffer` was allocated from does not support `VK_QUEUE_GRAPHICS_BIT`, for each element of `pRegions`, the `aspectMask` member of `imageSubresource` **must** not be `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`
- VUID-vkCmdCopyImageToBuffer-pRegions-00183
`dstBuffer` **must** be large enough to contain all buffer locations that are accessed according to `Buffer and Image Addressing`, for each element of `pRegions`
- VUID-vkCmdCopyImageToBuffer-pRegions-00184
The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-vkCmdCopyImageToBuffer-srcImage-00186
`srcImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-vkCmdCopyImageToBuffer-srcImage-01998
The `formatFeatures` of `srcImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_SRC_BIT`
- VUID-vkCmdCopyImageToBuffer-dstBuffer-00191
`dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdCopyImageToBuffer-dstBuffer-00192
If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdCopyImageToBuffer-srcImageLayout-00189
`srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-vkCmdCopyImageToBuffer-srcImageLayout-01397
`srcImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL`, or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-vkCmdCopyImageToBuffer-srcImage-07979
If `srcImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `imageOffset.y` **must** be 0 and `imageExtent.height` **must** be 1
- VUID-vkCmdCopyImageToBuffer-imageOffset-09104
For each element of `pRegions`, `imageOffset.z` and `(imageExtent.depth + imageOffset.z)` **must** both be greater than or equal to 0 and less than or equal to the depth of the specified `imageSubresource` of `srcImage`
- VUID-vkCmdCopyImageToBuffer-srcImage-07980
If `srcImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `imageOffset.z` **must** be 0 and `imageExtent.depth` **must** be 1
- VUID-vkCmdCopyImageToBuffer-srcImage-07274
For each element of `pRegions`, `imageOffset.x` **must** be a multiple of the `texel block extent width` of the `VkFormat` of `srcImage`
- VUID-vkCmdCopyImageToBuffer-srcImage-07275
For each element of `pRegions`, `imageOffset.y` **must** be a multiple of the `texel block extent height` of the `VkFormat` of `srcImage`
- VUID-vkCmdCopyImageToBuffer-srcImage-07276
For each element of `pRegions`, `imageOffset.z` **must** be a multiple of the `texel block extent depth` of the `VkFormat` of `srcImage`
- VUID-vkCmdCopyImageToBuffer-srcImage-00207
For each element of `pRegions`, if the sum of `imageOffset.x` and `extent.width` does not equal the width of the subresource specified by `imageSubresource`, `extent.width` **must** be a multiple of the `texel block extent width` of the `VkFormat` of `srcImage`
- VUID-vkCmdCopyImageToBuffer-srcImage-00208
For each element of `pRegions`, if the sum of `imageOffset.y` and `extent.height` does not equal the height of the subresource specified by `imageSubresource`, `extent.height` **must** be a multiple of the `texel block extent height` of the `VkFormat` of `srcImage`
- VUID-vkCmdCopyImageToBuffer-srcImage-00209
For each element of `pRegions`, if the sum of `imageOffset.z` and `extent.depth` does not equal the depth of the subresource specified by `srcSubresource`, `extent.depth` **must** be a multiple of the `texel block extent depth` of the `VkFormat` of `srcImage`
- VUID-vkCmdCopyImageToBuffer-imageSubresource-09105
For each element of `pRegions`, `imageSubresource.aspectMask` **must** specify aspects present in `srcImage`
- VUID-vkCmdCopyImageToBuffer-srcImage-07981
If `srcImage` has a `multi-planar format`, then for each element of `pRegions`, `imageSubresource.aspectMask` **must** be a single valid `multi-planar aspect mask` bit

- VUID-vkCmdCopyImageToBuffer-srcImage-07983
If `srcImage` is of type `VK_IMAGE_TYPE_3D`, for each element of `pRegions`, `imageSubresource.baseArrayLayer` **must** be 0 and `imageSubresource.layerCount` **must** be 1
- VUID-vkCmdCopyImageToBuffer-bufferRowLength-09106
For each element of `pRegions`, `bufferRowLength` **must** be a multiple of the `texel block extent width` of the `VkFormat` of `srcImage`
- VUID-vkCmdCopyImageToBuffer-bufferImageHeight-09107
For each element of `pRegions`, `bufferImageHeight` **must** be a multiple of the `texel block extent height` of the `VkFormat` of `srcImage`
- VUID-vkCmdCopyImageToBuffer-bufferRowLength-09108
For each element of `pRegions`, `bufferRowLength` divided by the `texel block extent width` and then multiplied by the texel block size of `srcImage` **must** be less than or equal to $2^{31}-1$
- VUID-vkCmdCopyImageToBuffer-srcImage-07975
If `srcImage` does not have either a depth/stencil format or a `multi-planar format`, then for each element of `pRegions`, `bufferOffset` **must** be a multiple of the `texel block size`
- VUID-vkCmdCopyImageToBuffer-srcImage-07976
If `srcImage` has a `multi-planar format`, then for each element of `pRegions`, `bufferOffset` **must** be a multiple of the element size of the compatible format for the format and the `aspectMask` of the `imageSubresource` as defined in `Compatible Formats of Planes of Multi-Planar Formats`
- VUID-vkCmdCopyImageToBuffer-srcImage-07978
If `srcImage` has a depth/stencil format, the `bufferOffset` member of any element of `pRegions` **must** be a multiple of 4

Valid Usage (Implicit)

- VUID-vkCmdCopyImageToBuffer-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdCopyImageToBuffer-srcImage-parameter
`srcImage` **must** be a valid `VkImage` handle
- VUID-vkCmdCopyImageToBuffer-srcImageLayout-parameter
`srcImageLayout` **must** be a valid `VkImageLayout` value
- VUID-vkCmdCopyImageToBuffer-dstBuffer-parameter
`dstBuffer` **must** be a valid `VkBuffer` handle
- VUID-vkCmdCopyImageToBuffer-pRegions-parameter
`pRegions` **must** be a valid pointer to an array of `regionCount` valid `VkBufferImageCopy` structures
- VUID-vkCmdCopyImageToBuffer-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdCopyImageToBuffer-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support

VK_QUEUE_COMPUTE_BIT, VK_QUEUE_GRAPHICS_BIT, or VK_QUEUE_TRANSFER_BIT operations

- VUID-vkCmdCopyImageToBuffer-renderpass

This command **must** only be called outside of a render pass instance

- VUID-vkCmdCopyImageToBuffer-regionCount-arraylength

regionCount **must** be greater than 0

- VUID-vkCmdCopyImageToBuffer-commonparent

Each of commandBuffer, dstBuffer, and srcImage **must** have been created, allocated, or retrieved from the same VkDevice

Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

For both vkCmdCopyBufferToImage and vkCmdCopyImageToBuffer, each element of pRegions is a structure defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkBufferImageCopy {
    VkDeviceSize          bufferOffset;
    uint32_t              bufferRowLength;
    uint32_t              bufferImageHeight;
    VkImageSubresourceLayers imageSubresource;
    VkOffset3D            imageOffset;
    VkExtent3D            imageExtent;
} VkBufferImageCopy;
```

- bufferOffset is the offset in bytes from the start of the buffer object where the image data is copied from or to.
- bufferRowLength and bufferImageHeight specify in texels a subregion of a larger two- or three-

dimensional image in buffer memory, and control the addressing calculations. If either of these values is zero, that aspect of the buffer memory is considered to be tightly packed according to the `imageExtent`.

- `imageSubresource` is a `VkImageSubresourceLayers` used to specify the specific image subresources of the image used for the source or destination image data.
- `imageOffset` selects the initial `x`, `y`, `z` offsets in texels of the sub-region of the source or destination image data.
- `imageExtent` is the size in texels of the image to copy in `width`, `height` and `depth`.

Valid Usage

- VUID-VkBufferImageCopy-bufferRowLength-09101
`bufferRowLength` **must** be 0, or greater than or equal to the `width` member of `imageExtent`
- VUID-VkBufferImageCopy-bufferImageHeight-09102
`bufferImageHeight` **must** be 0, or greater than or equal to the `height` member of `imageExtent`
- VUID-VkBufferImageCopy-aspectMask-09103
The `aspectMask` member of `imageSubresource` **must** only have a single bit set
- VUID-VkBufferImageCopy-imageExtent-06659
`imageExtent.width` **must** not be 0
- VUID-VkBufferImageCopy-imageExtent-06660
`imageExtent.height` **must** not be 0
- VUID-VkBufferImageCopy-imageExtent-06661
`imageExtent.depth` **must** not be 0

Valid Usage (Implicit)

- VUID-VkBufferImageCopy-imageSubresource-parameter
`imageSubresource` **must** be a valid `VkImageSubresourceLayers` structure

More extensible versions of the commands to copy between buffers and images are defined below.

To copy data from a buffer object to an image object, call:

```
// Provided by VK_KHR_copy_commands2
void vkCmdCopyBufferToImage2KHR(
    VkCommandBuffer                      commandBuffer,
    const VkCopyBufferToImageInfo2*     pCopyBufferToImageInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pCopyBufferToImageInfo` is a pointer to a `VkCopyBufferToImageInfo2` structure describing the copy parameters.

This command is functionally identical to [vkCmdCopyBufferToImage](#), but includes extensible sub-structures that include [sType](#) and [pNext](#) parameters, allowing them to be more easily extended.

Valid Usage

- VUID-vkCmdCopyBufferToImage2-commandBuffer-01828
If [commandBuffer](#) is an unprotected command buffer and [protectedNoFault](#) is not supported, [srcBuffer](#) **must** not be a protected buffer
- VUID-vkCmdCopyBufferToImage2-commandBuffer-01829
If [commandBuffer](#) is an unprotected command buffer and [protectedNoFault](#) is not supported, [dstImage](#) **must** not be a protected image
- VUID-vkCmdCopyBufferToImage2-commandBuffer-01830
If [commandBuffer](#) is a protected command buffer and [protectedNoFault](#) is not supported, [dstImage](#) **must** not be an unprotected image
- VUID-vkCmdCopyBufferToImage2-commandBuffer-07737
If the queue family used to create the [VkCommandPool](#) which [commandBuffer](#) was allocated from does not support [VK_QUEUE_GRAPHICS_BIT](#) or [VK_QUEUE_COMPUTE_BIT](#), the [bufferOffset](#) member of any element of [pCopyBufferToImageInfo->pRegions](#) **must** be a multiple of 4
- VUID-vkCmdCopyBufferToImage2-imageOffset-07738
The [imageOffset](#) and [imageExtent](#) members of each element of [pCopyBufferToImageInfo->pRegions](#) **must** respect the image transfer granularity requirements of [commandBuffer](#)'s command pool's queue family, as described in [VkQueueFamilyProperties](#)
- VUID-vkCmdCopyBufferToImage2-commandBuffer-07739
If the queue family used to create the [VkCommandPool](#) which [commandBuffer](#) was allocated from does not support [VK_QUEUE_GRAPHICS_BIT](#), for each element of [pCopyBufferToImageInfo->pRegions](#), the [aspectMask](#) member of [imageSubresource](#) **must** not be [VK_IMAGE_ASPECT_DEPTH_BIT](#) or [VK_IMAGE_ASPECT_STENCIL_BIT](#)

Valid Usage (Implicit)

- VUID-vkCmdCopyBufferToImage2-commandBuffer-parameter
[commandBuffer](#) **must** be a valid [VkCommandBuffer](#) handle
- VUID-vkCmdCopyBufferToImage2-pCopyBufferToImageInfo-parameter
[pCopyBufferToImageInfo](#) **must** be a valid pointer to a valid [VkCopyBufferToImageInfo2](#) structure
- VUID-vkCmdCopyBufferToImage2-commandBuffer-recording
[commandBuffer](#) **must** be in the [recording state](#)
- VUID-vkCmdCopyBufferToImage2-commandBuffer-cmdpool
The [VkCommandPool](#) that [commandBuffer](#) was allocated from **must** support [VK_QUEUE_COMPUTE_BIT](#), [VK_QUEUE_GRAPHICS_BIT](#), or [VK_QUEUE_TRANSFER_BIT](#) operations
- VUID-vkCmdCopyBufferToImage2-renderpass
This command **must** only be called outside of a render pass instance

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

The `VkCopyBufferToImageInfo2` structure is defined as:

```
typedef struct VkCopyBufferToImageInfo2 {  
    VkStructureType      sType;  
    const void*          pNext;  
    VkBuffer              srcBuffer;  
    VkImage               dstImage;  
    VkImageLayout         dstImageLayout;  
    uint32_t              regionCount;  
    const VkBufferImageCopy2* pRegions;  
} VkCopyBufferToImageInfo2;
```

```
// Provided by VK_KHR_copy_commands2  
typedef VkCopyBufferToImageInfo2 VkCopyBufferToImageInfo2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcBuffer` is the source buffer.
- `dstImage` is the destination image.
- `dstImageLayout` is the layout of the destination image subresources for the copy.
- `regionCount` is the number of regions to copy.
- `pRegions` is a pointer to an array of `VkBufferImageCopy2` structures specifying the regions to copy.

Valid Usage

- VUID-VkCopyBufferToImageInfo2-pRegions-04565
The image region specified by each element of **pRegions** **must** be contained within the specified **imageSubresource** of **dstImage**
- VUID-VkCopyBufferToImageInfo2-pRegions-00171
srcBuffer **must** be large enough to contain all buffer locations that are accessed according to **Buffer and Image Addressing**, for each element of **pRegions**
- VUID-VkCopyBufferToImageInfo2-pRegions-00173
The union of all source regions, and the union of all destination regions, specified by the elements of **pRegions**, **must** not overlap in memory
- VUID-VkCopyBufferToImageInfo2-srcBuffer-00174
srcBuffer **must** have been created with **VK_BUFFER_USAGE_TRANSFER_SRC_BIT** usage flag
- VUID-VkCopyBufferToImageInfo2-dstImage-01997
The **format features** of **dstImage** **must** contain **VK_FORMAT_FEATURE_TRANSFER_DST_BIT**
- VUID-VkCopyBufferToImageInfo2-srcBuffer-00176
If **srcBuffer** is non-sparse then it **must** be bound completely and contiguously to a single **VkDeviceMemory** object
- VUID-VkCopyBufferToImageInfo2-dstImage-00177
dstImage **must** have been created with **VK_IMAGE_USAGE_TRANSFER_DST_BIT** usage flag
- VUID-VkCopyBufferToImageInfo2-dstImageLayout-00180
dstImageLayout **must** specify the layout of the image subresources of **dstImage** specified in **pRegions** at the time this command is executed on a **VkDevice**
- VUID-VkCopyBufferToImageInfo2-dstImageLayout-01396
dstImageLayout **must** be **VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR**, **VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL**, or **VK_IMAGE_LAYOUT_GENERAL**
- VUID-VkCopyBufferToImageInfo2-pRegions-07931
If **VK_EXT_depth_range_unrestricted** is not enabled, for each element of **pRegions** whose **imageSubresource** contains a depth aspect, the data in **srcBuffer** **must** be in the range [0,1]
- VUID-VkCopyBufferToImageInfo2-dstImage-07966
If **dstImage** is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single **VkDeviceMemory** object
- VUID-VkCopyBufferToImageInfo2-imageSubresource-07967
The **imageSubresource.mipLevel** member of each element of **pRegions** **must** be less than the **mipLevels** specified in **VkImageCreateInfo** when **dstImage** was created
- VUID-VkCopyBufferToImageInfo2-imageSubresource-07968
imageSubresource.baseArrayLayer + **imageSubresource.layerCount** of each element of **pRegions** **must** be less than or equal to the **arrayLayers** specified in **VkImageCreateInfo** when **dstImage** was created
- VUID-VkCopyBufferToImageInfo2-dstImage-07973
dstImage **must** have a sample count equal to **VK_SAMPLE_COUNT_1_BIT**

- VUID-VkCopyBufferToImageInfo2-dstImage-07979
If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `imageOffset.y` **must** be 0 and `imageExtent.height` **must** be 1
- VUID-VkCopyBufferToImageInfo2-imageOffset-09104
For each element of `pRegions`, `imageOffset.z` and `(imageExtent.depth + imageOffset.z)` **must** both be greater than or equal to 0 and less than or equal to the depth of the specified `imageSubresource` of `dstImage`
- VUID-VkCopyBufferToImageInfo2-dstImage-07980
If `dstImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `imageOffset.z` **must** be 0 and `imageExtent.depth` **must** be 1
- VUID-VkCopyBufferToImageInfo2-dstImage-07274
For each element of `pRegions`, `imageOffset.x` **must** be a multiple of the `texel block extent width` of the `VkFormat` of `dstImage`
- VUID-VkCopyBufferToImageInfo2-dstImage-07275
For each element of `pRegions`, `imageOffset.y` **must** be a multiple of the `texel block extent height` of the `VkFormat` of `dstImage`
- VUID-VkCopyBufferToImageInfo2-dstImage-07276
For each element of `pRegions`, `imageOffset.z` **must** be a multiple of the `texel block extent depth` of the `VkFormat` of `dstImage`
- VUID-VkCopyBufferToImageInfo2-dstImage-00207
For each element of `pRegions`, if the sum of `imageOffset.x` and `extent.width` does not equal the width of the subresource specified by `imageSubresource`, `extent.width` **must** be a multiple of the `texel block extent width` of the `VkFormat` of `dstImage`
- VUID-VkCopyBufferToImageInfo2-dstImage-00208
For each element of `pRegions`, if the sum of `imageOffset.y` and `extent.height` does not equal the height of the subresource specified by `imageSubresource`, `extent.height` **must** be a multiple of the `texel block extent height` of the `VkFormat` of `dstImage`
- VUID-VkCopyBufferToImageInfo2-dstImage-00209
For each element of `pRegions`, if the sum of `imageOffset.z` and `extent.depth` does not equal the depth of the subresource specified by `srcSubresource`, `extent.depth` **must** be a multiple of the `texel block extent depth` of the `VkFormat` of `dstImage`
- VUID-VkCopyBufferToImageInfo2-imageSubresource-09105
For each element of `pRegions`, `imageSubresource.aspectMask` **must** specify aspects present in `dstImage`
- VUID-VkCopyBufferToImageInfo2-dstImage-07981
If `dstImage` has a `multi-planar format`, then for each element of `pRegions`, `imageSubresource.aspectMask` **must** be a single valid `multi-planar aspect mask` bit
- VUID-VkCopyBufferToImageInfo2-dstImage-07983
If `dstImage` is of type `VK_IMAGE_TYPE_3D`, for each element of `pRegions`, `imageSubresource.baseArrayLayer` **must** be 0 and `imageSubresource.layerCount` **must** be 1
- VUID-VkCopyBufferToImageInfo2-bufferRowLength-09106
For each element of `pRegions`, `bufferRowLength` **must** be a multiple of the `texel block extent width` of the `VkFormat` of `dstImage`

- VUID-VkCopyBufferToImageInfo2-bufferImageHeight-09107
For each element of `pRegions`, `bufferImageHeight` **must** be a multiple of the `texel block extent height` of the `VkFormat` of `dstImage`
- VUID-VkCopyBufferToImageInfo2-bufferRowLength-09108
For each element of `pRegions`, `bufferRowLength` divided by the `texel block extent width` and then multiplied by the texel block size of `dstImage` **must** be less than or equal to $2^{31}-1$
- VUID-VkCopyBufferToImageInfo2-dstImage-07975
If `dstImage` does not have either a depth/stencil format or a `multi-planar format`, then for each element of `pRegions`, `bufferOffset` **must** be a multiple of the `texel block size`
- VUID-VkCopyBufferToImageInfo2-dstImage-07976
If `dstImage` has a `multi-planar format`, then for each element of `pRegions`, `bufferOffset` **must** be a multiple of the element size of the compatible format for the format and the `aspectMask` of the `imageSubresource` as defined in `Compatible Formats of Planes of Multi-Planar Formats`
- VUID-VkCopyBufferToImageInfo2-dstImage-07978
If `dstImage` has a depth/stencil format, the `bufferOffset` member of any element of `pRegions` **must** be a multiple of 4
- VUID-VkCopyBufferToImageInfo2-pRegions-06223
For each element of `pRegions` not containing `VkCopyCommandTransformInfoQCOM` in its `pNext` chain, `imageOffset.x` and `(imageExtent.width + imageOffset.x)` **must** both be greater than or equal to 0 and less than or equal to the width of the specified `imageSubresource` of `dstImage`
- VUID-VkCopyBufferToImageInfo2-pRegions-06224
For each element of `pRegions` not containing `VkCopyCommandTransformInfoQCOM` in its `pNext` chain, `imageOffset.y` and `(imageExtent.height + imageOffset.y)` **must** both be greater than or equal to 0 and less than or equal to the height of the specified `imageSubresource` of `dstImage`

Valid Usage (Implicit)

- VUID-VkCopyBufferToImageInfo2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_COPY_BUFFER_TO_IMAGE_INFO_2`
- VUID-VkCopyBufferToImageInfo2-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkCopyBufferToImageInfo2-srcBuffer-parameter
`srcBuffer` **must** be a valid `VkBuffer` handle
- VUID-VkCopyBufferToImageInfo2-dstImage-parameter
`dstImage` **must** be a valid `VkImage` handle
- VUID-VkCopyBufferToImageInfo2-dstImageLayout-parameter
`dstImageLayout` **must** be a valid `VkImageLayout` value
- VUID-VkCopyBufferToImageInfo2-pRegions-parameter
`pRegions` **must** be a valid pointer to an array of `regionCount` valid `VkBufferImageCopy2` structures

- VUID-VkCopyBufferToImageInfo2-regionCount-arraylength
`regionCount` **must** be greater than 0
- VUID-VkCopyBufferToImageInfo2-commonparent
Both of `dstImage`, and `srcBuffer` **must** have been created, allocated, or retrieved from the same `VkDevice`

To copy data from an image object to a buffer object, call:

```
// Provided by VK_KHR_copy_commands2
void vkCmdCopyImageToBuffer2KHR(
    VkCommandBuffer                      commandBuffer,
    const VkCopyImageToBufferInfo2*     pCopyImageToBufferInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pCopyImageToBufferInfo` is a pointer to a `VkCopyImageToBufferInfo2` structure describing the copy parameters.

This command is functionally identical to `vkCmdCopyImageToBuffer`, but includes extensible sub-structures that include `sType` and `pNext` parameters, allowing them to be more easily extended.

Valid Usage

- VUID-vkCmdCopyImageToBuffer2-commandBuffer-01831
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image
- VUID-vkCmdCopyImageToBuffer2-commandBuffer-01832
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstBuffer` **must** not be a protected buffer
- VUID-vkCmdCopyImageToBuffer2-commandBuffer-01833
If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstBuffer` **must** not be an unprotected buffer
- VUID-vkCmdCopyImageToBuffer2-commandBuffer-07746
If the queue family used to create the `VkCommandPool` which `commandBuffer` was allocated from does not support `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`, the `bufferOffset` member of any element of `pCopyImageToBufferInfo->pRegions` **must** be a multiple of 4
- VUID-vkCmdCopyImageToBuffer2-imageOffset-07747
The `imageOffset` and `imageExtent` members of each element of `pCopyImageToBufferInfo->pRegions` **must** respect the image transfer granularity requirements of `commandBuffer`'s command pool's queue family, as described in `VkQueueFamilyProperties`
- VUID-vkCmdCopyImageToBuffer2-commandBuffer-10216
If the queue family used to create the `VkCommandPool` which `commandBuffer` was allocated from does not support `VK_QUEUE_GRAPHICS_BIT`, for each element of `pCopyImageToBufferInfo->pRegions`, the `aspectMask` member of `imageSubresource` **must** not be `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`

Valid Usage (Implicit)

- VUID-vkCmdCopyImageToBuffer2-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdCopyImageToBuffer2-pCopyImageToBufferInfo-parameter
`pCopyImageToBufferInfo` **must** be a valid pointer to a valid `VkCopyImageToBufferInfo2` structure
- VUID-vkCmdCopyImageToBuffer2-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdCopyImageToBuffer2-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations
- VUID-vkCmdCopyImageToBuffer2-renderpass
This command **must** only be called outside of a render pass instance

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

The `VkCopyImageToBufferInfo2` structure is defined as:

```
typedef struct VkCopyImageToBufferInfo2 {
    VkStructureType      sType;
    const void*          pNext;
    VkImage               srcImage;
    VkImageLayout         srcImageLayout;
    VkBuffer              dstBuffer;
    uint32_t              regionCount;
    const VkBufferImageCopy2* pRegions;
} VkCopyImageToBufferInfo2;
```

```
// Provided by VK_KHR_copy_commands2
typedef VkCopyImageToBufferInfo2 VkCopyImageToBufferInfo2KHR;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **srcImage** is the source image.
- **srcImageLayout** is the layout of the source image subresources for the copy.
- **dstBuffer** is the destination buffer.
- **regionCount** is the number of regions to copy.
- **pRegions** is a pointer to an array of **VkBufferImageCopy2** structures specifying the regions to copy.

Valid Usage

- VUID-VkCopyImageToBufferInfo2-pRegions-04566
The image region specified by each element of **pRegions** **must** be contained within the specified **imageSubresource** of **srcImage**
- VUID-VkCopyImageToBufferInfo2-pRegions-00183
dstBuffer **must** be large enough to contain all buffer locations that are accessed according to **Buffer and Image Addressing**, for each element of **pRegions**
- VUID-VkCopyImageToBufferInfo2-pRegions-00184
The union of all source regions, and the union of all destination regions, specified by the elements of **pRegions**, **must** not overlap in memory
- VUID-VkCopyImageToBufferInfo2-srcImage-00186
srcImage **must** have been created with **VK_IMAGE_USAGE_TRANSFER_SRC_BIT** usage flag
- VUID-VkCopyImageToBufferInfo2-srcImage-01998
The **format features** of **srcImage** **must** contain **VK_FORMAT_FEATURE_TRANSFER_SRC_BIT**
- VUID-VkCopyImageToBufferInfo2-dstBuffer-00191
dstBuffer **must** have been created with **VK_BUFFER_USAGE_TRANSFER_DST_BIT** usage flag
- VUID-VkCopyImageToBufferInfo2-dstBuffer-00192
If **dstBuffer** is non-sparse then it **must** be bound completely and contiguously to a single

VkDeviceMemory object

- VUID-VkCopyImageToBufferInfo2-srcImageLayout-00189
`srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-VkCopyImageToBufferInfo2-srcImageLayout-01397
`srcImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL`, or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-VkCopyImageToBufferInfo2-srcImage-07966
If `srcImage` is non-sparse then the image or the specified *disjoint* plane **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkCopyImageToBufferInfo2-imageSubresource-07967
The `imageSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in `VkImageCreateInfo` when `srcImage` was created
- VUID-VkCopyImageToBufferInfo2-imageSubresource-07968
`imageSubresource.baseArrayLayer` + `imageSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `srcImage` was created
- VUID-VkCopyImageToBufferInfo2-srcImage-07973
`srcImage` **must** have a sample count equal to `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkCopyImageToBufferInfo2-srcImage-07979
If `srcImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `imageOffset.y` **must** be 0 and `imageExtent.height` **must** be 1
- VUID-VkCopyImageToBufferInfo2-imageOffset-09104
For each element of `pRegions`, `imageOffset.z` and `(imageExtent.depth + imageOffset.z)` **must** both be greater than or equal to 0 and less than or equal to the depth of the specified `imageSubresource` of `srcImage`
- VUID-VkCopyImageToBufferInfo2-srcImage-07980
If `srcImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `imageOffset.z` **must** be 0 and `imageExtent.depth` **must** be 1
- VUID-VkCopyImageToBufferInfo2-srcImage-07274
For each element of `pRegions`, `imageOffset.x` **must** be a multiple of the `texel block extent width` of the `VkFormat` of `srcImage`
- VUID-VkCopyImageToBufferInfo2-srcImage-07275
For each element of `pRegions`, `imageOffset.y` **must** be a multiple of the `texel block extent height` of the `VkFormat` of `srcImage`
- VUID-VkCopyImageToBufferInfo2-srcImage-07276
For each element of `pRegions`, `imageOffset.z` **must** be a multiple of the `texel block extent depth` of the `VkFormat` of `srcImage`
- VUID-VkCopyImageToBufferInfo2-srcImage-00207
For each element of `pRegions`, if the sum of `imageOffset.x` and `extent.width` does not equal the width of the subresource specified by `imageSubresource`, `extent.width` **must** be a

multiple of the [texel block extent width](#) of the [VkFormat](#) of [srcImage](#)

- VUID-VkCopyImageToBufferInfo2-srcImage-00208

For each element of [pRegions](#), if the sum of [imageOffset.y](#) and [extent.height](#) does not equal the height of the subresource specified by [imageSubresource](#), [extent.height](#) **must** be a multiple of the [texel block extent height](#) of the [VkFormat](#) of [srcImage](#)

- VUID-VkCopyImageToBufferInfo2-srcImage-00209

For each element of [pRegions](#), if the sum of [imageOffset.z](#) and [extent.depth](#) does not equal the depth of the subresource specified by [srcSubresource](#), [extent.depth](#) **must** be a multiple of the [texel block extent depth](#) of the [VkFormat](#) of [srcImage](#)

- VUID-VkCopyImageToBufferInfo2-imageSubresource-09105

For each element of [pRegions](#), [imageSubresource.aspectMask](#) **must** specify aspects present in [srcImage](#)

- VUID-VkCopyImageToBufferInfo2-srcImage-07981

If [srcImage](#) has a [multi-planar format](#), then for each element of [pRegions](#), [imageSubresource.aspectMask](#) **must** be a single valid [multi-planar aspect mask](#) bit

- VUID-VkCopyImageToBufferInfo2-srcImage-07983

If [srcImage](#) is of type [VK_IMAGE_TYPE_3D](#), for each element of [pRegions](#), [imageSubresource.baseArrayLayer](#) **must** be 0 and [imageSubresource.layerCount](#) **must** be 1

- VUID-VkCopyImageToBufferInfo2-bufferRowLength-09106

For each element of [pRegions](#), [bufferRowLength](#) **must** be a multiple of the [texel block extent width](#) of the [VkFormat](#) of [srcImage](#)

- VUID-VkCopyImageToBufferInfo2-bufferImageHeight-09107

For each element of [pRegions](#), [bufferImageHeight](#) **must** be a multiple of the [texel block extent height](#) of the [VkFormat](#) of [srcImage](#)

- VUID-VkCopyImageToBufferInfo2-bufferRowLength-09108

For each element of [pRegions](#), [bufferRowLength](#) divided by the [texel block extent width](#) and then multiplied by the texel block size of [srcImage](#) **must** be less than or equal to $2^{31}-1$

- VUID-VkCopyImageToBufferInfo2-srcImage-07975

If [srcImage](#) does not have either a depth/stencil format or a [multi-planar format](#), then for each element of [pRegions](#), [bufferOffset](#) **must** be a multiple of the [texel block size](#)

- VUID-VkCopyImageToBufferInfo2-srcImage-07976

If [srcImage](#) has a [multi-planar format](#), then for each element of [pRegions](#), [bufferOffset](#) **must** be a multiple of the element size of the compatible format for the format and the [aspectMask](#) of the [imageSubresource](#) as defined in [Compatible Formats of Planes of Multi-Planar Formats](#)

- VUID-VkCopyImageToBufferInfo2-srcImage-07978

If [srcImage](#) has a depth/stencil format, the [bufferOffset](#) member of any element of [pRegions](#) **must** be a multiple of 4

- VUID-VkCopyImageToBufferInfo2-imageOffset-00197

For each element of [pRegions](#) not containing [VkCopyCommandTransformInfoQCOM](#) in its [pNext](#) chain, [imageOffset.x](#) and ([imageExtent.width](#) + [imageOffset.x](#)) **must** both be greater than or equal to 0 and less than or equal to the width of the specified [imageSubresource](#) of [srcImage](#)

- VUID-VkCopyImageToBufferInfo2-imageOffset-00198
For each element of `pRegions` not containing `VkCopyCommandTransformInfoQCOM` in its `pNext` chain, `imageOffset.y` and `(imageExtent.height + imageOffset.y)` **must** both be greater than or equal to `0` and less than or equal to the height of the specified `imageSubresource` of `srcImage`

Valid Usage (Implicit)

- VUID-VkCopyImageToBufferInfo2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_COPY_IMAGE_TO_BUFFER_INFO_2`
- VUID-VkCopyImageToBufferInfo2-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkCopyImageToBufferInfo2-srcImage-parameter
`srcImage` **must** be a valid `VkImage` handle
- VUID-VkCopyImageToBufferInfo2-srcImageLayout-parameter
`srcImageLayout` **must** be a valid `VkImageLayout` value
- VUID-VkCopyImageToBufferInfo2-dstBuffer-parameter
`dstBuffer` **must** be a valid `VkBuffer` handle
- VUID-VkCopyImageToBufferInfo2-pRegions-parameter
`pRegions` **must** be a valid pointer to an array of `regionCount` valid `VkBufferImageCopy2` structures
- VUID-VkCopyImageToBufferInfo2-regionCount-arraylength
`regionCount` **must** be greater than `0`
- VUID-VkCopyImageToBufferInfo2-commonparent
Both of `dstBuffer`, and `srcImage` **must** have been created, allocated, or retrieved from the same `VkDevice`

For both `vkCmdCopyBufferToImage2KHR` and `vkCmdCopyImageToBuffer2KHR`, each element of `pRegions` is a structure defined as:

```
typedef struct VkBufferImageCopy2 {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceSize        bufferOffset;
    uint32_t            bufferRowLength;
    uint32_t            bufferImageHeight;
    VkImageSubresourceLayers imageSubresource;
    VkOffset3D          imageOffset;
    VkExtent3D          imageExtent;
} VkBufferImageCopy2;
```

```
// Provided by VK_KHR_copy_commands2
typedef VkBufferImageCopy2 VkBufferImageCopy2KHR;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **bufferOffset** is the offset in bytes from the start of the buffer object where the image data is copied from or to.
- **bufferRowLength** and **bufferImageHeight** specify in texels a subregion of a larger two- or three-dimensional image in buffer memory, and control the addressing calculations. If either of these values is zero, that aspect of the buffer memory is considered to be tightly packed according to the **imageExtent**.
- **imageSubresource** is a [VkImageSubresourceLayers](#) used to specify the specific image subresources of the image used for the source or destination image data.
- **imageOffset** selects the initial **x**, **y**, **z** offsets in texels of the sub-region of the source or destination image data.
- **imageExtent** is the size in texels of the image to copy in **width**, **height** and **depth**.

This structure is functionally identical to [VkBufferImageCopy](#), but adds **sType** and **pNext** parameters, allowing it to be more easily extended.

Valid Usage

- VUID-VkBufferImageCopy2-bufferRowLength-09101
bufferRowLength **must** be 0, or greater than or equal to the **width** member of **imageExtent**
- VUID-VkBufferImageCopy2-bufferImageHeight-09102
bufferImageHeight **must** be 0, or greater than or equal to the **height** member of **imageExtent**
- VUID-VkBufferImageCopy2-aspectMask-09103
The **aspectMask** member of **imageSubresource** **must** only have a single bit set
- VUID-VkBufferImageCopy2-imageExtent-06659
imageExtent.width **must** not be 0
- VUID-VkBufferImageCopy2-imageExtent-06660
imageExtent.height **must** not be 0
- VUID-VkBufferImageCopy2-imageExtent-06661
imageExtent.depth **must** not be 0

Valid Usage (Implicit)

- VUID-VkBufferImageCopy2-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_BUFFER_IMAGE_COPY_2**
- VUID-VkBufferImageCopy2-pNext-pNext
pNext **must** be **NULL**

- VUID-VkBufferImageCopy2-imageSubresource-parameter
`imageSubresource` **must** be a valid [VkImageSubresourceLayers](#) structure

20.4. Image Copies With Scaling

To copy regions of a source image into a destination image, potentially performing format conversion, arbitrary scaling, and filtering, call:

```
// Provided by VK_VERSION_1_0
void vkCmdBlitImage(
    VkCommandBuffer          commandBuffer,
    VkImage                  srcImage,
    VkImageLayout            srcImageLayout,
    VkImage                  dstImage,
    VkImageLayout            dstImageLayout,
    uint32_t                 regionCount,
    const VkImageBlit*       pRegions,
    VkFilter                  filter);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `srcImage` is the source image.
- `srcImageLayout` is the layout of the source image subresources for the blit.
- `dstImage` is the destination image.
- `dstImageLayout` is the layout of the destination image subresources for the blit.
- `regionCount` is the number of regions to blit.
- `pRegions` is a pointer to an array of [VkImageBlit](#) structures specifying the regions to blit.
- `filter` is a [VkFilter](#) specifying the filter to apply if the blits require scaling.

`vkCmdBlitImage` **must** not be used for multisampled source or destination images. Use [vkCmdResolveImage](#) for this purpose.

As the sizes of the source and destination extents **can** differ in any dimension, texels in the source extent are scaled and filtered to the destination extent. Scaling occurs via the following operations:

- For each destination texel, the integer coordinate of that texel is converted to an unnormalized texture coordinate, using the effective inverse of the equations described in [unnormalized to integer conversion](#):

$$u_{\text{base}} = i + \frac{1}{2}$$

$$v_{\text{base}} = j + \frac{1}{2}$$

$$w_{\text{base}} = k + \frac{1}{2}$$

- These base coordinates are then offset by the first destination offset:

$$u_{\text{offset}} = u_{\text{base}} - x_{\text{dst0}}$$

$$v_{\text{offset}} = v_{\text{base}} - y_{\text{dst0}}$$

$$w_{\text{offset}} = w_{\text{base}} - z_{\text{dst0}}$$

$$a_{\text{offset}} = a - \text{baseArrayCount}_{\text{dst}}$$

- The scale is determined from the source and destination regions, and applied to the offset coordinates:

$$\text{scale}_u = (x_{\text{src1}} - x_{\text{src0}}) / (x_{\text{dst1}} - x_{\text{dst0}})$$

$$\text{scale}_v = (y_{\text{src1}} - y_{\text{src0}}) / (y_{\text{dst1}} - y_{\text{dst0}})$$

$$\text{scale}_w = (z_{\text{src1}} - z_{\text{src0}}) / (z_{\text{dst1}} - z_{\text{dst0}})$$

$$u_{\text{scaled}} = u_{\text{offset}} \times \text{scale}_u$$

$$v_{\text{scaled}} = v_{\text{offset}} \times \text{scale}_v$$

$$w_{\text{scaled}} = w_{\text{offset}} \times \text{scale}_w$$

- Finally the source offset is added to the scaled coordinates, to determine the final unnormalized coordinates used to sample from `srcImage`:

$$u = u_{\text{scaled}} + x_{\text{src0}}$$

$$v = v_{\text{scaled}} + y_{\text{src0}}$$

$$w = w_{\text{scaled}} + z_{\text{src0}}$$

$q = \text{mipLevel}$

$a = a_{\text{offset}} + \text{baseArrayCount}_{\text{src}}$

These coordinates are used to sample from the source image, as described in [Image Operations chapter](#), with the filter mode equal to that of `filter`, a mipmap mode of `VK_SAMPLER_MIPMAP_MODE_NEAREST` and an address mode of `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE`. Implementations **must** clamp at the edge of the source image, and **may** additionally clamp to the edge of the source region.

NOTE

Due to allowable rounding errors in the generation of the source texture coordinates, it is not always possible to guarantee exactly which source texels will be sampled for a given blit. As rounding errors are implementation-dependent, the exact results of a blitting operation are also implementation-dependent.

Blits are done layer by layer starting with the `baseArrayLayer` member of `srcSubresource` for the source and `dstSubresource` for the destination. `layerCount` layers are blitted to the destination image.

When blitting 3D textures, slices in the destination region bounded by `dstOffsets[0].z` and `dstOffsets[1].z` are sampled from slices in the source region bounded by `srcOffsets[0].z` and `srcOffsets[1].z`. If the `filter` parameter is `VK_FILTER_LINEAR` then the value sampled from the source image is taken by doing linear filtering using the interpolated `z` coordinate represented by `w` in the previous equations. If the `filter` parameter is `VK_FILTER_NEAREST` then the value sampled from the source image is taken from the single nearest slice, with an implementation-dependent arithmetic rounding mode.

The following filtering and conversion rules apply:

- Integer formats **can** only be converted to other integer formats with the same signedness.
- No format conversion is supported between depth/stencil images. The formats **must** match.
- Format conversions on unorm, snorm, scaled and packed float formats of the copied aspect of the image are performed by first converting the pixels to float values.
- For sRGB source formats, nonlinear RGB values are converted to linear representation prior to filtering.
- After filtering, the float values are first clamped and then cast to the destination image format. In case of sRGB destination format, linear RGB values are converted to nonlinear representation before writing the pixel to the image.

Signed and unsigned integers are converted by first clamping to the representable range of the destination format, then casting the value.

Valid Usage

- VUID-vkCmdBlitImage-commandBuffer-01834

If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported,

`srcImage` **must** not be a protected image

- VUID-vkCmdBlitImage-commandBuffer-01835

If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image

- VUID-vkCmdBlitImage-commandBuffer-01836

If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image

- VUID-vkCmdBlitImage-pRegions-00217

The union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory with any texel that **may** be sampled during the blit operation

- VUID-vkCmdBlitImage-srcImage-01999

The `format features` of `srcImage` **must** contain `VK_FORMAT_FEATURE_BLIT_SRC_BIT`

- VUID-vkCmdBlitImage-srcImage-06421

`srcImage` **must** not use a `format that requires a sampler Y'CBCR conversion`

- VUID-vkCmdBlitImage-srcImage-00219

`srcImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` usage flag

- VUID-vkCmdBlitImage-srcImage-00220

If `srcImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdBlitImage-srcImageLayout-00221

`srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`

- VUID-vkCmdBlitImage-srcImageLayout-01398

`srcImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`

- VUID-vkCmdBlitImage-srcImage-09459

If `srcImage` and `dstImage` are the same, and an elements of `pRegions` contains the `srcSubresource` and `dstSubresource` with matching `mipLevel` and overlapping array layers, then the `srcImageLayout` and `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_GENERAL` or `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`

- VUID-vkCmdBlitImage-dstImage-02000

The `format features` of `dstImage` **must** contain `VK_FORMAT_FEATURE_BLIT_DST_BIT`

- VUID-vkCmdBlitImage-dstImage-06422

`dstImage` **must** not use a `format that requires a sampler Y'CBCR conversion`

- VUID-vkCmdBlitImage-dstImage-00224

`dstImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag

- VUID-vkCmdBlitImage-dstImage-00225

If `dstImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdBlitImage-dstImageLayout-00226

`dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in

`pRegions` at the time this command is executed on a `VkDevice`

- VUID-vkCmdBlitImage-dstImageLayout-01399
`dstImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-vkCmdBlitImage-srcImage-00229
If either of `srcImage` or `dstImage` was created with a signed integer `VkFormat`, the other **must** also have been created with a signed integer `VkFormat`
- VUID-vkCmdBlitImage-srcImage-00230
If either of `srcImage` or `dstImage` was created with an unsigned integer `VkFormat`, the other **must** also have been created with an unsigned integer `VkFormat`
- VUID-vkCmdBlitImage-srcImage-00231
If either of `srcImage` or `dstImage` was created with a depth/stencil format, the other **must** have exactly the same format
- VUID-vkCmdBlitImage-srcImage-00232
If `srcImage` was created with a depth/stencil format, `filter` **must** be `VK_FILTER_NEAREST`
- VUID-vkCmdBlitImage-srcImage-00233
`srcImage` **must** have been created with a `samples` value of `VK_SAMPLE_COUNT_1_BIT`
- VUID-vkCmdBlitImage-dstImage-00234
`dstImage` **must** have been created with a `samples` value of `VK_SAMPLE_COUNT_1_BIT`
- VUID-vkCmdBlitImage-filter-02001
If `filter` is `VK_FILTER_LINEAR`, then the `format features` of `srcImage` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdBlitImage-filter-02002
If `filter` is `VK_FILTER_CUBIC_EXT`, then the `format features` of `srcImage` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT`
- VUID-vkCmdBlitImage-filter-00237
If `filter` is `VK_FILTER_CUBIC_EXT`, `srcImage` **must** be of type `VK_IMAGE_TYPE_2D`
- VUID-vkCmdBlitImage-srcSubresource-01705
The `srcSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in `VkImageCreateInfo` when `srcImage` was created
- VUID-vkCmdBlitImage-dstSubresource-01706
The `dstSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in `VkImageCreateInfo` when `dstImage` was created
- VUID-vkCmdBlitImage-srcSubresource-01707
`srcSubresource.baseArrayLayer` + `srcSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `srcImage` was created
- VUID-vkCmdBlitImage-dstSubresource-01708
`dstSubresource.baseArrayLayer` + `dstSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `dstImage` was created
- VUID-vkCmdBlitImage-srcImage-00240

If either `srcImage` or `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `srcSubresource.baseArrayLayer` and `dstSubresource.baseArrayLayer` **must** each be `0`, and `srcSubresource.layerCount` and `dstSubresource.layerCount` **must** each be `1`

- VUID-vkCmdBlitImage-aspectMask-00241

For each element of `pRegions`, `srcSubresource.aspectMask` **must** specify aspects present in `srcImage`

- VUID-vkCmdBlitImage-aspectMask-00242

For each element of `pRegions`, `dstSubresource.aspectMask` **must** specify aspects present in `dstImage`

- VUID-vkCmdBlitImage-srcOffset-00243

For each element of `pRegions`, `srcOffsets[0].x` and `srcOffsets[1].x` **must** both be greater than or equal to `0` and less than or equal to the width of the specified `srcSubresource` of `srcImage`

- VUID-vkCmdBlitImage-srcOffset-00244

For each element of `pRegions`, `srcOffsets[0].y` and `srcOffsets[1].y` **must** both be greater than or equal to `0` and less than or equal to the height of the specified `srcSubresource` of `srcImage`

- VUID-vkCmdBlitImage-srcImage-00245

If `srcImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `srcOffsets[0].y` **must** be `0` and `srcOffsets[1].y` **must** be `1`

- VUID-vkCmdBlitImage-srcOffset-00246

For each element of `pRegions`, `srcOffsets[0].z` and `srcOffsets[1].z` **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `srcSubresource` of `srcImage`

- VUID-vkCmdBlitImage-srcImage-00247

If `srcImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `srcOffsets[0].z` **must** be `0` and `srcOffsets[1].z` **must** be `1`

- VUID-vkCmdBlitImage-dstOffset-00248

For each element of `pRegions`, `dstOffsets[0].x` and `dstOffsets[1].x` **must** both be greater than or equal to `0` and less than or equal to the width of the specified `dstSubresource` of `dstImage`

- VUID-vkCmdBlitImage-dstOffset-00249

For each element of `pRegions`, `dstOffsets[0].y` and `dstOffsets[1].y` **must** both be greater than or equal to `0` and less than or equal to the height of the specified `dstSubresource` of `dstImage`

- VUID-vkCmdBlitImage-dstImage-00250

If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `dstOffsets[0].y` **must** be `0` and `dstOffsets[1].y` **must** be `1`

- VUID-vkCmdBlitImage-dstOffset-00251

For each element of `pRegions`, `dstOffsets[0].z` and `dstOffsets[1].z` **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `dstSubresource` of `dstImage`

- VUID-vkCmdBlitImage-dstImage-00252

If `dstImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `dstOffsets[0].z` **must** be `0` and `dstOffsets[1].z` **must** be `1`

Valid Usage (Implicit)

- VUID-vkCmdBlitImage-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdBlitImage-srcImage-parameter
`srcImage` **must** be a valid `VkImage` handle
- VUID-vkCmdBlitImage-srcImageLayout-parameter
`srcImageLayout` **must** be a valid `VkImageLayout` value
- VUID-vkCmdBlitImage-dstImage-parameter
`dstImage` **must** be a valid `VkImage` handle
- VUID-vkCmdBlitImage-dstImageLayout-parameter
`dstImageLayout` **must** be a valid `VkImageLayout` value
- VUID-vkCmdBlitImage-pRegions-parameter
`pRegions` **must** be a valid pointer to an array of `regionCount` valid `VkImageBlit` structures
- VUID-vkCmdBlitImage-filter-parameter
`filter` **must** be a valid `VkFilter` value
- VUID-vkCmdBlitImage-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdBlitImage-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdBlitImage-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdBlitImage-regionCount-arraylength
`regionCount` **must** be greater than `0`
- VUID-vkCmdBlitImage-commonparent
Each of `commandBuffer`, `dstImage`, and `srcImage` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_GRAPHICS_BIT	Action

The `VkImageBlit` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageBlit {
    VkImageSubresourceLayers    srcSubresource;
    VkOffset3D                  srcOffsets[2];
    VkImageSubresourceLayers    dstSubresource;
    VkOffset3D                  dstOffsets[2];
} VkImageBlit;
```

- `srcSubresource` is the subresource to blit from.
- `srcOffsets` is a pointer to an array of two `VkOffset3D` structures specifying the bounds of the source region within `srcSubresource`.
- `dstSubresource` is the subresource to blit into.
- `dstOffsets` is a pointer to an array of two `VkOffset3D` structures specifying the bounds of the destination region within `dstSubresource`.

For each element of the `pRegions` array, a blit operation is performed for the specified source and destination regions.

Valid Usage

- VUID-VkImageBlit-aspectMask-00238
The `aspectMask` member of `srcSubresource` and `dstSubresource` **must** match
- VUID-VkImageBlit-layerCount-08800
The `layerCount` members of `srcSubresource` or `dstSubresource` **must** match

Valid Usage (Implicit)

- VUID-VkImageBlit-srcSubresource-parameter
`srcSubresource` **must** be a valid `VkImageSubresourceLayers` structure
- VUID-VkImageBlit-dstSubresource-parameter
`dstSubresource` **must** be a valid `VkImageSubresourceLayers` structure

A more extensible version of the blit image command is defined below.

To copy regions of a source image into a destination image, potentially performing format conversion, arbitrary scaling, and filtering, call:

```
// Provided by VK_KHR_copy_commands2
void vkCmdBlitImage2KHR(
    VkCommandBuffer                      commandBuffer,
    const VkBlitImageInfo2*             pBlitImageInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pBlitImageInfo` is a pointer to a `VkBlitImageInfo2` structure describing the blit parameters.

This command is functionally identical to `vkCmdBlitImage`, but includes extensible sub-structures that include `sType` and `pNext` parameters, allowing them to be more easily extended.

Valid Usage

- VUID-vkCmdBlitImage2-commandBuffer-01834
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image
- VUID-vkCmdBlitImage2-commandBuffer-01835
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image
- VUID-vkCmdBlitImage2-commandBuffer-01836
If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image

Valid Usage (Implicit)

- VUID-vkCmdBlitImage2-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdBlitImage2-pBlitImageInfo-parameter
`pBlitImageInfo` **must** be a valid pointer to a valid `VkBlitImageInfo2` structure
- VUID-vkCmdBlitImage2-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdBlitImage2-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdBlitImage2-renderpass
This command **must** only be called outside of a render pass instance

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_GRAPHICS_BIT	Action

The `VkBlitImageInfo2` structure is defined as:

```
typedef struct VkBlitImageInfo2 {  
    VkStructureType    sType;  
    const void*        pNext;  
    VkImage             srcImage;  
    VkImageLayout      srcImageLayout;  
    VkImage             dstImage;  
    VkImageLayout      dstImageLayout;  
    uint32_t           regionCount;  
    const VkImageBlit2* pRegions;  
    VkFilter            filter;  
} VkBlitImageInfo2;
```

```
// Provided by VK_KHR_copy_commands2  
typedef VkBlitImageInfo2 VkBlitImageInfo2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcImage` is the source image.
- `srcImageLayout` is the layout of the source image subresources for the blit.
- `dstImage` is the destination image.
- `dstImageLayout` is the layout of the destination image subresources for the blit.
- `regionCount` is the number of regions to blit.
- `pRegions` is a pointer to an array of `VkImageBlit2` structures specifying the regions to blit.
- `filter` is a `VkFilter` specifying the filter to apply if the blits require scaling.

Valid Usage

- VUID-VkBlitImageInfo2-pRegions-00217
The union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory with any texel that **may** be sampled during the blit operation
- VUID-VkBlitImageInfo2-srcImage-01999
The `format features` of `srcImage` **must** contain `VK_FORMAT_FEATURE_BLIT_SRC_BIT`
- VUID-VkBlitImageInfo2-srcImage-06421
`srcImage` **must** not use a `format that requires a sampler Y'CBCR conversion`
- VUID-VkBlitImageInfo2-srcImage-00219
`srcImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-VkBlitImageInfo2-srcImage-00220
If `srcImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkBlitImageInfo2-srcImageLayout-00221
`srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-VkBlitImageInfo2-srcImageLayout-01398
`srcImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-VkBlitImageInfo2-srcImage-09459
If `srcImage` and `dstImage` are the same, and an elements of `pRegions` contains the `srcSubresource` and `dstSubresource` with matching `mipLevel` and overlapping array layers, then the `srcImageLayout` and `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_GENERAL` or `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`
- VUID-VkBlitImageInfo2-dstImage-02000
The `format features` of `dstImage` **must** contain `VK_FORMAT_FEATURE_BLIT_DST_BIT`
- VUID-VkBlitImageInfo2-dstImage-06422
`dstImage` **must** not use a `format that requires a sampler Y'CBCR conversion`
- VUID-VkBlitImageInfo2-dstImage-00224
`dstImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-VkBlitImageInfo2-dstImage-00225
If `dstImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkBlitImageInfo2-dstImageLayout-00226
`dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-VkBlitImageInfo2-dstImageLayout-01399
`dstImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-VkBlitImageInfo2-srcImage-00229
If either of `srcImage` or `dstImage` was created with a signed integer `VkFormat`, the other

must also have been created with a signed integer `VkFormat`

- VUID-VkBlitImageInfo2-srcImage-00230

If either of `srcImage` or `dstImage` was created with an unsigned integer `VkFormat`, the other **must** also have been created with an unsigned integer `VkFormat`

- VUID-VkBlitImageInfo2-srcImage-00231

If either of `srcImage` or `dstImage` was created with a depth/stencil format, the other **must** have exactly the same format

- VUID-VkBlitImageInfo2-srcImage-00232

If `srcImage` was created with a depth/stencil format, `filter` **must** be `VK_FILTER_NEAREST`

- VUID-VkBlitImageInfo2-srcImage-00233

`srcImage` **must** have been created with a `samples` value of `VK_SAMPLE_COUNT_1_BIT`

- VUID-VkBlitImageInfo2-dstImage-00234

`dstImage` **must** have been created with a `samples` value of `VK_SAMPLE_COUNT_1_BIT`

- VUID-VkBlitImageInfo2-filter-02001

If `filter` is `VK_FILTER_LINEAR`, then the `format features` of `srcImage` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-VkBlitImageInfo2-filter-02002

If `filter` is `VK_FILTER_CUBIC_EXT`, then the `format features` of `srcImage` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT`

- VUID-VkBlitImageInfo2-filter-00237

If `filter` is `VK_FILTER_CUBIC_EXT`, `srcImage` **must** be of type `VK_IMAGE_TYPE_2D`

- VUID-VkBlitImageInfo2-srcSubresource-01705

The `srcSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in `VkImageCreateInfo` when `srcImage` was created

- VUID-VkBlitImageInfo2-dstSubresource-01706

The `dstSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in `VkImageCreateInfo` when `dstImage` was created

- VUID-VkBlitImageInfo2-srcSubresource-01707

`srcSubresource.baseArrayLayer` + `srcSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `srcImage` was created

- VUID-VkBlitImageInfo2-dstSubresource-01708

`dstSubresource.baseArrayLayer` + `dstSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `dstImage` was created

- VUID-VkBlitImageInfo2-srcImage-00240

If either `srcImage` or `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `srcSubresource.baseArrayLayer` and `dstSubresource.baseArrayLayer` **must** each be 0, and `srcSubresource.layerCount` and `dstSubresource.layerCount` **must** each be 1

- VUID-VkBlitImageInfo2-aspectMask-00241

For each element of `pRegions`, `srcSubresource.aspectMask` **must** specify aspects present in `srcImage`

- VUID-VkBlitImageInfo2-aspectMask-00242
For each element of `pRegions`, `dstSubresource.aspectMask` **must** specify aspects present in `dstImage`
- VUID-VkBlitImageInfo2-srcOffset-00243
For each element of `pRegions`, `srcOffsets[0].x` and `srcOffsets[1].x` **must** both be greater than or equal to `0` and less than or equal to the width of the specified `srcSubresource` of `srcImage`
- VUID-VkBlitImageInfo2-srcOffset-00244
For each element of `pRegions`, `srcOffsets[0].y` and `srcOffsets[1].y` **must** both be greater than or equal to `0` and less than or equal to the height of the specified `srcSubresource` of `srcImage`
- VUID-VkBlitImageInfo2-srcImage-00245
If `srcImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `srcOffsets[0].y` **must** be `0` and `srcOffsets[1].y` **must** be `1`
- VUID-VkBlitImageInfo2-srcOffset-00246
For each element of `pRegions`, `srcOffsets[0].z` and `srcOffsets[1].z` **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `srcSubresource` of `srcImage`
- VUID-VkBlitImageInfo2-srcImage-00247
If `srcImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `srcOffsets[0].z` **must** be `0` and `srcOffsets[1].z` **must** be `1`
- VUID-VkBlitImageInfo2-dstOffset-00248
For each element of `pRegions`, `dstOffsets[0].x` and `dstOffsets[1].x` **must** both be greater than or equal to `0` and less than or equal to the width of the specified `dstSubresource` of `dstImage`
- VUID-VkBlitImageInfo2-dstOffset-00249
For each element of `pRegions`, `dstOffsets[0].y` and `dstOffsets[1].y` **must** both be greater than or equal to `0` and less than or equal to the height of the specified `dstSubresource` of `dstImage`
- VUID-VkBlitImageInfo2-dstImage-00250
If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `dstOffsets[0].y` **must** be `0` and `dstOffsets[1].y` **must** be `1`
- VUID-VkBlitImageInfo2-dstOffset-00251
For each element of `pRegions`, `dstOffsets[0].z` and `dstOffsets[1].z` **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `dstSubresource` of `dstImage`
- VUID-VkBlitImageInfo2-dstImage-00252
If `dstImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `dstOffsets[0].z` **must** be `0` and `dstOffsets[1].z` **must** be `1`

Valid Usage (Implicit)

- VUID-VkBlitImageInfo2-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_BLIT_IMAGE_INFO_2`

- `VUID-VkBlitImageInfo2-pNext-pNext`
`pNext` **must** be `NULL`
- `VUID-VkBlitImageInfo2-srcImage-parameter`
`srcImage` **must** be a valid `VkImage` handle
- `VUID-VkBlitImageInfo2-srcImageLayout-parameter`
`srcImageLayout` **must** be a valid `VkImageLayout` value
- `VUID-VkBlitImageInfo2-dstImage-parameter`
`dstImage` **must** be a valid `VkImage` handle
- `VUID-VkBlitImageInfo2-dstImageLayout-parameter`
`dstImageLayout` **must** be a valid `VkImageLayout` value
- `VUID-VkBlitImageInfo2-pRegions-parameter`
`pRegions` **must** be a valid pointer to an array of `regionCount` valid `VkImageBlit2` structures
- `VUID-VkBlitImageInfo2-filter-parameter`
`filter` **must** be a valid `VkFilter` value
- `VUID-VkBlitImageInfo2-regionCount-arraylength`
`regionCount` **must** be greater than `0`
- `VUID-VkBlitImageInfo2-commonparent`
Both of `dstImage`, and `srcImage` **must** have been created, allocated, or retrieved from the same `VkDevice`

The `VkImageBlit2` structure is defined as:

```
typedef struct VkImageBlit2 {  
    VkStructureType    sType;  
    const void*        pNext;  
    VkImageSubresourceLayers srcSubresource;  
    VkOffset3D          srcOffsets[2];  
    VkImageSubresourceLayers dstSubresource;  
    VkOffset3D          dstOffsets[2];  
} VkImageBlit2;
```

```
// Provided by VK_KHR_copy_commands2  
typedef VkImageBlit2 VkImageBlit2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcSubresource` is the subresource to blit from.
- `srcOffsets` is a pointer to an array of two `VkOffset3D` structures specifying the bounds of the source region within `srcSubresource`.
- `dstSubresource` is the subresource to blit into.

- `dstOffsets` is a pointer to an array of two `VkOffset3D` structures specifying the bounds of the destination region within `dstSubresource`.

For each element of the `pRegions` array, a blit operation is performed for the specified source and destination regions.

Valid Usage

- VUID-VkImageBlit2-aspectMask-00238
The `aspectMask` member of `srcSubresource` and `dstSubresource` **must** match
- VUID-VkImageBlit2-layerCount-08800
The `layerCount` members of `srcSubresource` or `dstSubresource` **must** match

Valid Usage (Implicit)

- VUID-VkImageBlit2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_BLIT_2`
- VUID-VkImageBlit2-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkImageBlit2-srcSubresource-parameter
`srcSubresource` **must** be a valid `VkImageSubresourceLayers` structure
- VUID-VkImageBlit2-dstSubresource-parameter
`dstSubresource` **must** be a valid `VkImageSubresourceLayers` structure

20.5. Resolving Multisample Images

To resolve a multisample color image to a non-multisample color image, call:

```
// Provided by VK_VERSION_1_0
void vkCmdResolveImage(
    VkCommandBuffer          commandBuffer,
    VkImage                  srcImage,
    VkImageLayout            srcImageLayout,
    VkImage                  dstImage,
    VkImageLayout            dstImageLayout,
    uint32_t                 regionCount,
    const VkImageResolve*    pRegions);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `srcImage` is the source image.
- `srcImageLayout` is the layout of the source image subresources for the resolve.
- `dstImage` is the destination image.

- `dstImageLayout` is the layout of the destination image subresources for the resolve.
- `regionCount` is the number of regions to resolve.
- `pRegions` is a pointer to an array of `VkImageResolve` structures specifying the regions to resolve.

During the resolve the samples corresponding to each pixel location in the source are converted to a single sample before being written to the destination.

If the source format is a floating-point or normalized type, the resolve mode is chosen as implementation-dependent behavior. If the resolve mode requires to calculate the result from multiple samples, such as by computing an average or weighted average of the samples, the values for each pixel are resolved with implementation-defined numerical precision.

If the `numeric format` of `srcImage` uses sRGB encoding and the resolve mode requires the implementation to convert the samples to floating-point to perform the calculations, the implementation **should** convert samples from nonlinear to linear before resolving the samples as described in the “sRGB EOTF” section of the [Khronos Data Format Specification](#). In this case, the implementation **must** convert the linear averaged value to nonlinear before writing the resolved result to `dstImage`.

If the source format is an integer type, a single sample’s value is selected for each pixel.

`srcOffset` and `dstOffset` select the initial `x`, `y`, and `z` offsets in texels of the sub-regions of the source and destination image data. `extent` is the size in texels of the source image to resolve in `width`, `height` and `depth`. Each element of `pRegions` **must** be a region that is contained within its corresponding image.

Resolves are done layer by layer starting with `baseArrayLayer` member of `srcSubresource` for the source and `dstSubresource` for the destination. `layerCount` layers are resolved to the destination image.

Valid Usage

- VUID-vkCmdResolveImage-commandBuffer-01837
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image
- VUID-vkCmdResolveImage-commandBuffer-01838
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image
- VUID-vkCmdResolveImage-commandBuffer-01839
If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image
- VUID-vkCmdResolveImage-pRegions-00255
The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-vkCmdResolveImage-srcImage-00256
If `srcImage` is non-sparse then it **must** be bound completely and contiguously to a single

VkDeviceMemory object

- VUID-vkCmdResolveImage-srcImage-00257
`srcImage` **must** have a sample count equal to any valid sample count value other than `VK_SAMPLE_COUNT_1_BIT`
- VUID-vkCmdResolveImage-dstImage-00258
If `dstImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdResolveImage-dstImage-00259
`dstImage` **must** have a sample count equal to `VK_SAMPLE_COUNT_1_BIT`
- VUID-vkCmdResolveImage-srcImageLayout-00260
`srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-vkCmdResolveImage-srcImageLayout-01400
`srcImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-vkCmdResolveImage-dstImageLayout-00262
`dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-vkCmdResolveImage-dstImageLayout-01401
`dstImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-vkCmdResolveImage-dstImage-02003
The `format features` of `dstImage` **must** contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`
- VUID-vkCmdResolveImage-srcImage-01386
`srcImage` and `dstImage` **must** have been created with the same image format
- VUID-vkCmdResolveImage-srcSubresource-01709
The `srcSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in `VkImageCreateInfo` when `srcImage` was created
- VUID-vkCmdResolveImage-dstSubresource-01710
The `dstSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in `VkImageCreateInfo` when `dstImage` was created
- VUID-vkCmdResolveImage-srcSubresource-01711
`srcSubresource.baseArrayLayer` + `srcSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `srcImage` was created
- VUID-vkCmdResolveImage-dstSubresource-01712
`dstSubresource.baseArrayLayer` + `dstSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `dstImage` was created
- VUID-vkCmdResolveImage-srcImage-04446
If `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `srcSubresource.layerCount` **must** be 1

- VUID-vkCmdResolveImage-srcImage-04447
If `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `dstSubresource.baseArrayLayer` **must** be `0` and `dstSubresource.layerCount` **must** be `1`
- VUID-vkCmdResolveImage-srcOffset-00269
For each element of `pRegions`, `srcOffset.x` and `(extent.width + srcOffset.x)` **must** both be greater than or equal to `0` and less than or equal to the width of the specified `srcSubresource` of `srcImage`
- VUID-vkCmdResolveImage-srcOffset-00270
For each element of `pRegions`, `srcOffset.y` and `(extent.height + srcOffset.y)` **must** both be greater than or equal to `0` and less than or equal to the height of the specified `srcSubresource` of `srcImage`
- VUID-vkCmdResolveImage-srcImage-00271
If `srcImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `srcOffset.y` **must** be `0` and `extent.height` **must** be `1`
- VUID-vkCmdResolveImage-srcOffset-00272
For each element of `pRegions`, `srcOffset.z` and `(extent.depth + srcOffset.z)` **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `srcSubresource` of `srcImage`
- VUID-vkCmdResolveImage-srcImage-00273
If `srcImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `srcOffset.z` **must** be `0` and `extent.depth` **must** be `1`
- VUID-vkCmdResolveImage-dstOffset-00274
For each element of `pRegions`, `dstOffset.x` and `(extent.width + dstOffset.x)` **must** both be greater than or equal to `0` and less than or equal to the width of the specified `dstSubresource` of `dstImage`
- VUID-vkCmdResolveImage-dstOffset-00275
For each element of `pRegions`, `dstOffset.y` and `(extent.height + dstOffset.y)` **must** both be greater than or equal to `0` and less than or equal to the height of the specified `dstSubresource` of `dstImage`
- VUID-vkCmdResolveImage-dstImage-00276
If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `dstOffset.y` **must** be `0` and `extent.height` **must** be `1`
- VUID-vkCmdResolveImage-dstOffset-00277
For each element of `pRegions`, `dstOffset.z` and `(extent.depth + dstOffset.z)` **must** both be greater than or equal to `0` and less than or equal to the depth of the specified `dstSubresource` of `dstImage`
- VUID-vkCmdResolveImage-dstImage-00278
If `dstImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `dstOffset.z` **must** be `0` and `extent.depth` **must** be `1`
- VUID-vkCmdResolveImage-srcImage-06762
`srcImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` usage flag
- VUID-vkCmdResolveImage-srcImage-06763
The `format features` of `srcImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_SRC_BIT`

- VUID-vkCmdResolveImage-dstImage-06764
`dstImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag
- VUID-vkCmdResolveImage-dstImage-06765
The `format features` of `dstImage` **must** contain `VK_FORMAT_FEATURE_TRANSFER_DST_BIT`
- VUID-vkCmdResolveImage-srcSubresource-11800
`srcSubresource.aspectMask` for each element in `pRegions` **must** not specify an aspect which is not part of the image format of `srcImage`
- VUID-vkCmdResolveImage-dstSubresource-11801
`dstSubresource.aspectMask` for each element in `pRegions` **must** not specify an aspect which is not part of the image format of `dstImage`
- VUID-vkCmdResolveImage-srcSubresource-11802
`srcSubresource.aspectMask` **must** equal `dstSubresource.aspectMask` for each element in `pRegions`

Valid Usage (Implicit)

- VUID-vkCmdResolveImage-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdResolveImage-srcImage-parameter
`srcImage` **must** be a valid `VkImage` handle
- VUID-vkCmdResolveImage-srcImageLayout-parameter
`srcImageLayout` **must** be a valid `VkImageLayout` value
- VUID-vkCmdResolveImage-dstImage-parameter
`dstImage` **must** be a valid `VkImage` handle
- VUID-vkCmdResolveImage-dstImageLayout-parameter
`dstImageLayout` **must** be a valid `VkImageLayout` value
- VUID-vkCmdResolveImage-pRegions-parameter
`pRegions` **must** be a valid pointer to an array of `regionCount` valid `VkImageResolve` structures
- VUID-vkCmdResolveImage-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdResolveImage-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdResolveImage-renderpass
This command **must** only be called outside of a render pass instance
- VUID-vkCmdResolveImage-regionCount-arraylength
`regionCount` **must** be greater than 0
- VUID-vkCmdResolveImage-commonparent
Each of `commandBuffer`, `dstImage`, and `srcImage` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_GRAPHICS_BIT	Action

The `VkImageResolve` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageResolve {
    VkImageSubresourceLayers    srcSubresource;
    VkOffset3D                  srcOffset;
    VkImageSubresourceLayers    dstSubresource;
    VkOffset3D                  dstOffset;
    VkExtent3D                  extent;
} VkImageResolve;
```

- `srcSubresource` and `dstSubresource` are `VkImageSubresourceLayers` structures specifying the image subresources of the images used for the source and destination image data, respectively. Resolve of depth/stencil images is not supported.
- `srcOffset` and `dstOffset` select the initial `x`, `y`, and `z` offsets in texels of the sub-regions of the source and destination image data.
- `extent` is the size in texels of the source image to resolve in `width`, `height` and `depth`.

Valid Usage

- VUID-VkImageResolve-aspectMask-10981
The `aspectMask` member of `srcSubresource` and `dstSubresource` **must** only contain `VK_IMAGE_ASPECT_COLOR_BIT`
- VUID-VkImageResolve-layerCount-08803
The `layerCount` member of `srcSubresource` and `dstSubresource` **must** match

Valid Usage (Implicit)

- VUID-VkImageResolve-srcSubresource-parameter
`srcSubresource` **must** be a valid [VkImageSubresourceLayers](#) structure
- VUID-VkImageResolve-dstSubresource-parameter
`dstSubresource` **must** be a valid [VkImageSubresourceLayers](#) structure

A more extensible version of the resolve image command is defined below.

To resolve a multisample image to a non-multisample image, call:

```
// Provided by VK_KHR_copy_commands2
void vkCmdResolveImage2KHR(
    VkCommandBuffer          commandBuffer,
    const VkResolveImageInfo2* pResolveImageInfo);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pResolveImageInfo` is a pointer to a [VkResolveImageInfo2](#) structure describing the resolve parameters.

This command is functionally identical to [vkCmdResolveImage](#), but includes extensible sub-structures that include `sType` and `pNext` parameters, allowing them to be more easily extended.

Valid Usage

- VUID-vkCmdResolveImage2-commandBuffer-01837
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `srcImage` **must** not be a protected image
- VUID-vkCmdResolveImage2-commandBuffer-01838
If `commandBuffer` is an unprotected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be a protected image
- VUID-vkCmdResolveImage2-commandBuffer-01839
If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, `dstImage` **must** not be an unprotected image

Valid Usage (Implicit)

- VUID-vkCmdResolveImage2-commandBuffer-parameter
`commandBuffer` **must** be a valid [VkCommandBuffer](#) handle
- VUID-vkCmdResolveImage2-pResolveImageInfo-parameter
`pResolveImageInfo` **must** be a valid pointer to a valid [VkResolveImageInfo2](#) structure
- VUID-vkCmdResolveImage2-commandBuffer-recording
`commandBuffer` **must** be in the [recording state](#)

- VUID-vkCmdResolveImage2-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdResolveImage2-renderpass
This command **must** only be called outside of a render pass instance

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_GRAPHICS_BIT	Action

The `VkResolveImageInfo2` structure is defined as:

```
typedef struct VkResolveImageInfo2 {
    VkStructureType    sType;
    const void*        pNext;
    VkImage             srcImage;
    VkImageLayout      srcImageLayout;
    VkImage             dstImage;
    VkImageLayout      dstImageLayout;
    uint32_t           regionCount;
    const VkImageResolve2* pRegions;
} VkResolveImageInfo2;
```

```
// Provided by VK_KHR_copy_commands2
typedef VkResolveImageInfo2 VkResolveImageInfo2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcImage` is the source image.
- `srcImageLayout` is the layout of the source image subresources for the resolve.
- `dstImage` is the destination image.

- `dstImageLayout` is the layout of the destination image subresources for the resolve.
- `regionCount` is the number of regions to resolve.
- `pRegions` is a pointer to an array of `VkImageResolve2` structures specifying the regions to resolve.

Valid Usage

- VUID-VkResolveImageInfo2-pRegions-00255
The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- VUID-VkResolveImageInfo2-srcImage-00256
If `srcImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkResolveImageInfo2-srcImage-00257
`srcImage` **must** have a sample count equal to any valid sample count value other than `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkResolveImageInfo2-dstImage-00258
If `dstImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-VkResolveImageInfo2-dstImage-00259
`dstImage` **must** have a sample count equal to `VK_SAMPLE_COUNT_1_BIT`
- VUID-VkResolveImageInfo2-srcImageLayout-00260
`srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-VkResolveImageInfo2-srcImageLayout-01400
`srcImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-VkResolveImageInfo2-dstImageLayout-00262
`dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- VUID-VkResolveImageInfo2-dstImageLayout-01401
`dstImageLayout` **must** be `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR`, `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- VUID-VkResolveImageInfo2-dstImage-02003
The `format features` of `dstImage` **must** contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`
- VUID-VkResolveImageInfo2-srcImage-01386
`srcImage` and `dstImage` **must** have been created with the same image format
- VUID-VkResolveImageInfo2-srcSubresource-01709
The `srcSubresource.mipLevel` member of each element of `pRegions` **must** be less than the `mipLevels` specified in `VkImageCreateInfo` when `srcImage` was created
- VUID-VkResolveImageInfo2-dstSubresource-01710
The `dstSubresource.mipLevel` member of each element of `pRegions` **must** be less than the

`mipLevels` specified in `VkImageCreateInfo` when `dstImage` was created

- VUID-VkResolveImageInfo2-srcSubresource-01711
`srcSubresource.baseArrayLayer` + `srcSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `srcImage` was created
- VUID-VkResolveImageInfo2-dstSubresource-01712
`dstSubresource.baseArrayLayer` + `dstSubresource.layerCount` of each element of `pRegions` **must** be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `dstImage` was created
- VUID-VkResolveImageInfo2-srcImage-04446
If `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `srcSubresource.layerCount` **must** be 1
- VUID-VkResolveImageInfo2-srcImage-04447
If `dstImage` is of type `VK_IMAGE_TYPE_3D`, then for each element of `pRegions`, `dstSubresource.baseArrayLayer` **must** be 0 and `dstSubresource.layerCount` **must** be 1
- VUID-VkResolveImageInfo2-srcOffset-00269
For each element of `pRegions`, `srcOffset.x` and `(extent.width + srcOffset.x)` **must** both be greater than or equal to 0 and less than or equal to the width of the specified `srcSubresource` of `srcImage`
- VUID-VkResolveImageInfo2-srcOffset-00270
For each element of `pRegions`, `srcOffset.y` and `(extent.height + srcOffset.y)` **must** both be greater than or equal to 0 and less than or equal to the height of the specified `srcSubresource` of `srcImage`
- VUID-VkResolveImageInfo2-srcImage-00271
If `srcImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `srcOffset.y` **must** be 0 and `extent.height` **must** be 1
- VUID-VkResolveImageInfo2-srcOffset-00272
For each element of `pRegions`, `srcOffset.z` and `(extent.depth + srcOffset.z)` **must** both be greater than or equal to 0 and less than or equal to the depth of the specified `srcSubresource` of `srcImage`
- VUID-VkResolveImageInfo2-srcImage-00273
If `srcImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then for each element of `pRegions`, `srcOffset.z` **must** be 0 and `extent.depth` **must** be 1
- VUID-VkResolveImageInfo2-dstOffset-00274
For each element of `pRegions`, `dstOffset.x` and `(extent.width + dstOffset.x)` **must** both be greater than or equal to 0 and less than or equal to the width of the specified `dstSubresource` of `dstImage`
- VUID-VkResolveImageInfo2-dstOffset-00275
For each element of `pRegions`, `dstOffset.y` and `(extent.height + dstOffset.y)` **must** both be greater than or equal to 0 and less than or equal to the height of the specified `dstSubresource` of `dstImage`
- VUID-VkResolveImageInfo2-dstImage-00276
If `dstImage` is of type `VK_IMAGE_TYPE_1D`, then for each element of `pRegions`, `dstOffset.y`

must be 0 and **extent.height** **must** be 1

- VUID-VkResolveImageInfo2-dstOffset-00277

For each element of **pRegions**, **dstOffset.z** and **(extent.depth + dstOffset.z)** **must** both be greater than or equal to 0 and less than or equal to the depth of the specified **dstSubresource** of **dstImage**

- VUID-VkResolveImageInfo2-dstImage-00278

If **dstImage** is of type **VK_IMAGE_TYPE_1D** or **VK_IMAGE_TYPE_2D**, then for each element of **pRegions**, **dstOffset.z** **must** be 0 and **extent.depth** **must** be 1

- VUID-VkResolveImageInfo2-srcImage-06762

srcImage **must** have been created with **VK_IMAGE_USAGE_TRANSFER_SRC_BIT** usage flag

- VUID-VkResolveImageInfo2-srcImage-06763

The **format features** of **srcImage** **must** contain **VK_FORMAT_FEATURE_TRANSFER_SRC_BIT**

- VUID-VkResolveImageInfo2-dstImage-06764

dstImage **must** have been created with **VK_IMAGE_USAGE_TRANSFER_DST_BIT** usage flag

- VUID-VkResolveImageInfo2-dstImage-06765

The **format features** of **dstImage** **must** contain **VK_FORMAT_FEATURE_TRANSFER_DST_BIT**

- VUID-VkResolveImageInfo2-srcSubresource-11800

srcSubresource.aspectMask for each element in **pRegions** **must** not specify an aspect which is not part of the image format of **srcImage**

- VUID-VkResolveImageInfo2-dstSubresource-11801

dstSubresource.aspectMask for each element in **pRegions** **must** not specify an aspect which is not part of the image format of **dstImage**

- VUID-VkResolveImageInfo2-srcSubresource-11802

srcSubresource.aspectMask **must** equal **dstSubresource.aspectMask** for each element in **pRegions**

Valid Usage (Implicit)

- VUID-VkResolveImageInfo2-sType-sType

sType **must** be **VK_STRUCTURE_TYPE_RESOLVE_IMAGE_INFO_2**

- VUID-VkResolveImageInfo2-pNext-pNext

pNext **must** be **NULL**

- VUID-VkResolveImageInfo2-srcImage-parameter

srcImage **must** be a valid **VkImage** handle

- VUID-VkResolveImageInfo2-srcImageLayout-parameter

srcImageLayout **must** be a valid **VkImageLayout** value

- VUID-VkResolveImageInfo2-dstImage-parameter

dstImage **must** be a valid **VkImage** handle

- VUID-VkResolveImageInfo2-dstImageLayout-parameter

dstImageLayout **must** be a valid **VkImageLayout** value

- VUID-VkResolveImageInfo2-pRegions-parameter

pRegions **must** be a valid pointer to an array of **regionCount** valid **VkImageResolve2** structures

- VUID-VkResolveImageInfo2-regionCount-arraylength
regionCount **must** be greater than 0
- VUID-VkResolveImageInfo2-commonparent
Both of **dstImage**, and **srcImage** **must** have been created, allocated, or retrieved from the same **VkDevice**

The **VkImageResolve2** structure is defined as:

```
typedef struct VkImageResolve2 {  
    VkStructureType      sType;  
    const void*          pNext;  
    VkImageSubresourceLayers srcSubresource;  
    VkOffset3D           srcOffset;  
    VkImageSubresourceLayers dstSubresource;  
    VkOffset3D           dstOffset;  
    VkExtent3D           extent;  
} VkImageResolve2;
```

```
// Provided by VK_KHR_copy_commands2  
typedef VkImageResolve2 VkImageResolve2KHR;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **srcSubresource** and **dstSubresource** are **VkImageSubresourceLayers** structures specifying the image subresources of the images used for the source and destination image data, respectively. Resolve of depth/stencil images is not supported.
- **srcOffset** and **dstOffset** select the initial **x**, **y**, and **z** offsets in texels of the sub-regions of the source and destination image data.
- **extent** is the size in texels of the source image to resolve in **width**, **height** and **depth**.

Valid Usage

- VUID-VkImageResolve2-aspectMask-10993
The **aspectMask** member of **srcSubresource** and **dstSubresource** **must** only contain **VK_IMAGE_ASPECT_COLOR_BIT**
- VUID-VkImageResolve2-layerCount-08803
The **layerCount** member of **srcSubresource** and **dstSubresource** **must** match

Valid Usage (Implicit)

- VUID-VkImageResolve2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_RESOLVE_2`
- VUID-VkImageResolve2-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkImageResolve2-srcSubresource-parameter
`srcSubresource` **must** be a valid `VkImageSubresourceLayers` structure
- VUID-VkImageResolve2-dstSubresource-parameter
`dstSubresource` **must** be a valid `VkImageSubresourceLayers` structure

20.6. Object Refreshes

Safety critical applications **may** need to contend with single event upsets (SEUs). For a Vulkan object explicitly backed by device memory, such as a `VkImage` or `VkBuffer`, an application **can** bind its backing memory to a SEU-safe heap with the `VK_MEMORY_HEAP_SEU_SAFE_BIT` bit set. Alternatively, an application **can** also periodically reload the non-SEU-safe device memory contents from a known SEU-safe portion of host memory, or otherwise periodically regenerate or refresh the contents of non-SEU-safe device memory.

However, an implementation **may** store implementation-specific internal object data in non-SEU-safe memory, and Base Vulkan provides no method to determine which object types this applies to or how to refresh their data. An application **can** query the list of object types that have implementation internal object data stored in non-SEU-safe memory using `vkGetPhysicalDeviceRefreshableObjectTypesKHR`, and **can** instruct the implementation to refresh the internal data of specific objects from a backup in SEU-safe memory using the `vkCmdRefreshObjectsKHR` command.

To refresh a list of objects as a pipelined operation, call:

```
// Provided by VK_KHR_object_refresh
void vkCmdRefreshObjectsKHR(
    VkCommandBuffer          commandBuffer,
    const VkRefreshObjectListKHR* pRefreshObjects);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pRefreshObjects` is a pointer to a `VkRefreshObjectListKHR` structure specifying the list of objects to refresh.

The access scope for object refreshes falls under the `VK_ACCESS_TRANSFER_WRITE_BIT`, and the pipeline stages for identifying the synchronization scope **must** include `VK_PIPELINE_STAGE_TRANSFER_BIT`.

NOTE

If an implementation does not store a supplied object's internal data in SEU-susceptible memory, it **may** ignore the refresh command for that object.

Valid Usage (Implicit)

- VUID-vkCmdRefreshObjectsKHR-commandBuffer-parameter `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdRefreshObjectsKHR-pRefreshObjects-parameter `pRefreshObjects` **must** be a valid pointer to a valid `VkRefreshObjectListKHR` structure
- VUID-vkCmdRefreshObjectsKHR-commandBuffer-recording `commandBuffer` **must** be in the `recording` state
- VUID-vkCmdRefreshObjectsKHR-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations
- VUID-vkCmdRefreshObjectsKHR-renderpass
This command **must** only be called outside of a render pass instance

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Outside	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

The `VkRefreshObjectListKHR` structure is defined as:

```
// Provided by VK_KHR_object_refresh
typedef struct VkRefreshObjectListKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           objectCount;
    const VkRefreshObjectKHR* pObjects;
} VkRefreshObjectListKHR;
```

- `sType` is a [VkStructureType](#) value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `objectCount` is the number of objects to refresh.
- `pObjects` is a pointer to an array of [VkRefreshObjectKHR](#) structures, defining the objects to refresh.

Valid Usage (Implicit)

- VUID-VkRefreshObjectListKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_REFRESH_OBJECT_LIST_KHR`
- VUID-VkRefreshObjectListKHR-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkRefreshObjectListKHR-pObjects-parameter
`pObjects` **must** be a valid pointer to an array of `objectCount` valid [VkRefreshObjectKHR](#) structures
- VUID-VkRefreshObjectListKHR-objectCount-arraylength
`objectCount` **must** be greater than `0`

The [VkRefreshObjectKHR](#) structure is defined as:

```
// Provided by VK_KHR_object_refresh
typedef struct VkRefreshObjectKHR {
    VkObjectType          objectType;
    uint64_t              objectHandle;
    VkRefreshObjectFlagsKHR flags;
} VkRefreshObjectKHR;
```

- `objectType` is a [VkObjectType](#) specifying the type of the object to refresh.
- `objectHandle` is the object to refresh.
- `flags` is a bitmask of [VkRefreshObjectFlagsKHR](#).

Valid Usage

- VUID-VkRefreshObjectKHR-objectHandle-05069
`objectHandle` **must** be a valid Vulkan handle of the type associated with `objectType` as defined in the [VkObjectType and Vulkan Handle Relationship](#) table
- VUID-VkRefreshObjectKHR-objectType-05070
`objectType` **must** not be `VK_OBJECT_TYPE_UNKNOWN`

Valid Usage (Implicit)

- VUID-VkRefreshObjectKHR-objectType-parameter
`objectType` **must** be a valid `VkObjectType` value
- VUID-VkRefreshObjectKHR-flags-zero bitmask
`flags` **must** be 0

Host Synchronization

- Host access to `objectHandle` **must** be externally synchronized

```
// Provided by VK_KHR_object_refresh
typedef enum VkRefreshObjectFlagBitsKHR {
} VkRefreshObjectFlagBitsKHR;
```

```
// Provided by VK_KHR_object_refresh
typedef VkFlags VkRefreshObjectFlagsKHR;
```

`VkRefreshObjectFlagsKHR` is a bitmask type for setting a mask, but is currently reserved for future use.

Chapter 21. Drawing Commands

Drawing commands (commands with **Draw** in the name) provoke work in a graphics pipeline. Drawing commands are recorded into a command buffer and when executed by a queue, will produce work which executes according to the bound graphics pipeline. A graphics pipeline **must** be bound to a command buffer before any drawing commands are recorded in that command buffer.

Each draw is made up of zero or more vertices and zero or more instances, which are processed by the device and result in the assembly of primitives. Primitives are assembled according to the `pInputAssemblyState` member of the `VkGraphicsPipelineCreateInfo` structure, which is of type `VkPipelineInputAssemblyStateCreateInfo`:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineInputAssemblyStateCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkPipelineInputAssemblyStateCreateFlags flags;
    VkPrimitiveTopology        topology;
    VkBool32                  primitiveRestartEnable;
} VkPipelineInputAssemblyStateCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `topology` is a `VkPrimitiveTopology` defining the primitive topology, as described below.
- `primitiveRestartEnable` controls whether a special vertex index value is treated as restarting the assembly of primitives. This enable only applies to indexed draws (`vkCmdDrawIndexed`, and `vkCmdDrawIndexedIndirect`), and the special index value is either `0xFFFFFFFF` when the `indexType` parameter of `vkCmdBindIndexBuffer` is equal to `VK_INDEX_TYPE_UINT32`; `0xFF` when `indexType` is equal to `VK_INDEX_TYPE_UINT8`; or `0xFFFF` when `indexType` is equal to `VK_INDEX_TYPE_UINT16`. Primitive restart is not allowed for “list” topologies.

Restarting the assembly of primitives discards the most recent index values if those elements formed an incomplete primitive, and restarts the primitive assembly using the subsequent indices, but only assembling the immediately following element through the end of the originally specified elements. The primitive restart index value comparison is performed before adding the `vertexOffset` value to the index value.

Valid Usage

- VUID-VkPipelineInputAssemblyStateCreateInfo-topology-06252
If `topology` is `VK_PRIMITIVE_TOPOLOGY_POINT_LIST`, `VK_PRIMITIVE_TOPOLOGY_LINE_LIST`, `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST`, `VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY`, or `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY`, `primitiveRestartEnable` **must** be

VK_FALSE

- VUID-VkPipelineInputAssemblyStateCreateInfo-topology-06253
If `topology` is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, `primitiveRestartEnable` **must** be `VK_FALSE`
- VUID-VkPipelineInputAssemblyStateCreateInfo-topology-00429
If the `geometryShader` feature is not enabled, `topology` **must** not be any of `VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY`,
`VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY`,
`VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY` or
`VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY`
- VUID-VkPipelineInputAssemblyStateCreateInfo-topology-00430
If the `tessellationShader` feature is not enabled, `topology` **must** not be `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`

Valid Usage (Implicit)

- VUID-VkPipelineInputAssemblyStateCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO`
- VUID-VkPipelineInputAssemblyStateCreateInfo-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkPipelineInputAssemblyStateCreateInfo-flags-zeroBitmask
`flags` **must** be `0`
- VUID-VkPipelineInputAssemblyStateCreateInfo-topology-parameter
`topology` **must** be a valid `VkPrimitiveTopology` value

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineInputAssemblyStateCreateFlags;
```

`VkPipelineInputAssemblyStateCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

To [dynamically control](#) whether a special vertex index value is treated as restarting the assembly of primitives, call:

```
// Provided by VK_EXT_extended_dynamic_state2
void vkCmdSetPrimitiveRestartEnableEXT(
    VkCommandBuffer          commandBuffer,
    VkBool32                  primitiveRestartEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `primitiveRestartEnable` controls whether a special vertex index value is treated as restarting the assembly of primitives. It behaves in the same way as `VkPipelineInputAssemblyStateCreateInfo::primitiveRestartEnable`

This command sets the primitive restart enable for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineInputAssemblyStateCreateInfo::primitiveRestartEnable` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetPrimitiveRestartEnable-None-08970

At least one of the following **must** be true:

- the `extendedDynamicState2` feature is enabled

Valid Usage (Implicit)

- VUID-vkCmdSetPrimitiveRestartEnable-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetPrimitiveRestartEnable-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetPrimitiveRestartEnable-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

21.1. Primitive Topologies

Primitive topology determines how consecutive vertices are organized into primitives, and determines the type of primitive that is used at the beginning of the graphics pipeline. The effective topology for later stages of the pipeline is altered by tessellation or geometry shading (if either is in use) and depends on the execution modes of those shaders.

The primitive topologies defined by [VkPrimitiveTopology](#) are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPrimitiveTopology {
    VK_PRIMITIVE_TOPOLOGY_POINT_LIST = 0,
    VK_PRIMITIVE_TOPOLOGY_LINE_LIST = 1,
    VK_PRIMITIVE_TOPOLOGY_LINE_STRIP = 2,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST = 3,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP = 4,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN = 5,
    VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY = 6,
    VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY = 7,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY = 8,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY = 9,
    VK_PRIMITIVE_TOPOLOGY_PATCH_LIST = 10,
} VkPrimitiveTopology;
```

- [VK_PRIMITIVE_TOPOLOGY_POINT_LIST](#) specifies a series of [separate point primitives](#).
- [VK_PRIMITIVE_TOPOLOGY_LINE_LIST](#) specifies a series of [separate line primitives](#).
- [VK_PRIMITIVE_TOPOLOGY_LINE_STRIP](#) specifies a series of [connected line primitives](#) with consecutive lines sharing a vertex.
- [VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST](#) specifies a series of [separate triangle primitives](#).
- [VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP](#) specifies a series of [connected triangle primitives](#) with consecutive triangles sharing an edge.
- [VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN](#) specifies a series of [connected triangle primitives](#) with all triangles sharing a common vertex.
- [VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY](#) specifies a series of [separate line primitives with adjacency](#).
- [VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY](#) specifies a series of [connected line primitives with adjacency](#), with consecutive primitives sharing three vertices.
- [VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY](#) specifies a series of [separate triangle primitives with adjacency](#).
- [VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY](#) specifies [connected triangle primitives with adjacency](#), with consecutive triangles sharing an edge.
- [VK_PRIMITIVE_TOPOLOGY_PATCH_LIST](#) specifies [separate patch primitives](#).

Each primitive topology, and its construction from a list of vertices, is described in detail below with a supporting diagram, according to the following key:

•	Vertex	A point in 3-dimensional space. Positions chosen within the diagrams are arbitrary and for illustration only.
5	Vertex Number	Sequence position of a vertex within the provided vertex data.

	Provoking Vertex	Provoking vertex within the main primitive. The tail is angled towards the relevant primitive. Used in flat shading .
	Primitive Edge	An edge connecting the points of a main primitive.
	Adjacency Edge	Points connected by these lines do not contribute to a main primitive, and are only accessible in a geometry shader .
	Winding Order	The relative order in which vertices are defined within a primitive, used in the facing determination . This ordering has no specific start or end point.

The diagrams are supported with mathematical definitions where the vertices (v) and primitives (p) are numbered starting from 0; v_0 is the first vertex in the provided data and p_0 is the first primitive in the set of primitives defined by the vertices and topology.

To [dynamically set](#) primitive topology, call:

```
// Provided by VK_EXT_extended_dynamic_state
void vkCmdSetPrimitiveTopologyEXT(
    VkCommandBuffer          commandBuffer,
    VkPrimitiveTopology       primitiveTopology);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `primitiveTopology` specifies the primitive topology to use for drawing.

This command sets the primitive topology for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineInputAssemblyStateCreateInfo::topology` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetPrimitiveTopology-None-08971
At least one of the following **must** be true:
 - the `extendedDynamicState` feature is enabled

Valid Usage (Implicit)

- VUID-vkCmdSetPrimitiveTopology-commandBuffer-parameter `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetPrimitiveTopology-primitiveTopology-parameter `primitiveTopology` **must** be a valid `VkPrimitiveTopology` value
- VUID-vkCmdSetPrimitiveTopology-commandBuffer-recording `commandBuffer` **must** be in the `recording state`

- VUID-vkCmdSetPrimitiveTopology-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

21.1.1. Topology Class

The primitive topologies are grouped into the following topology classes:

Table 25. Topology Classes

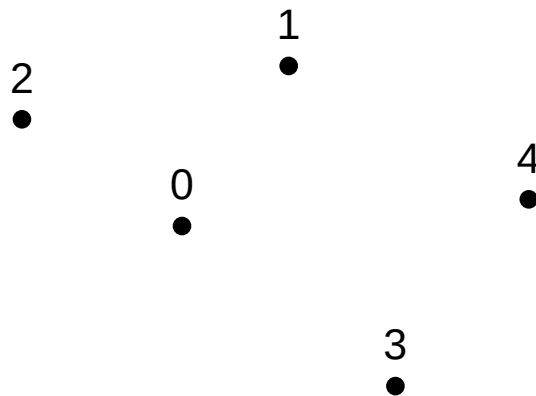
Topology Class	Primitive Topology
Point	<code>VK_PRIMITIVE_TOPOLOGY_POINT_LIST</code>
Line	<code>VK_PRIMITIVE_TOPOLOGY_LINE_LIST</code> , <code>VK_PRIMITIVE_TOPOLOGY_LINE_STRIP</code> , <code>VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY</code> , <code>VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY</code>
Triangle	<code>VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST</code> , <code>VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP</code> , <code>VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN</code> , <code>VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY</code> , <code>VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY</code>
Patch	<code>VK_PRIMITIVE_TOPOLOGY_PATCH_LIST</code>

21.1.2. Point Lists

When the topology is `VK_PRIMITIVE_TOPOLOGY_POINT_LIST`, each consecutive vertex defines a single point primitive, according to the equation:

$$p_i = \{v_i\}$$

As there is only one vertex, that vertex is the provoking vertex. The number of primitives generated is equal to `vertexCount`.



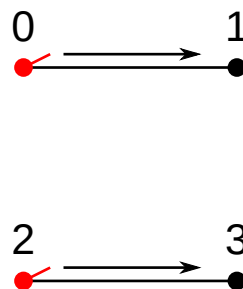
21.1.3. Line Lists

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_LINE_LIST`, each consecutive pair of vertices defines a single line primitive, according to the equation:

$$p_i = \{v_{2i}, v_{2i+1}\}$$

The number of primitives generated is equal to $\lceil \text{vertexCount} / 2 \rceil$.

The provoking vertex for p_i is v_{2i} .



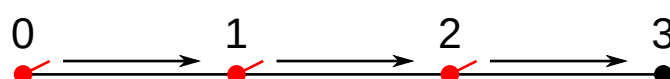
21.1.4. Line Strips

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_LINE_STRIP`, one line primitive is defined by each vertex and the following vertex, according to the equation:

$$p_i = \{v_i, v_{i+1}\}$$

The number of primitives generated is equal to $\max(0, \text{vertexCount} - 1)$.

The provoking vertex for p_i is v_i .



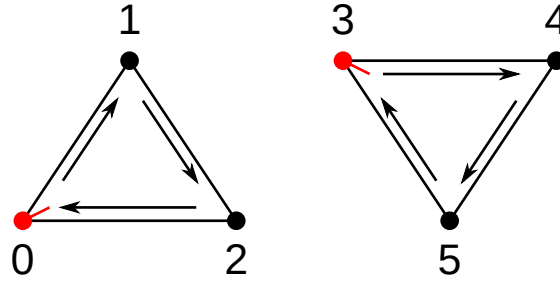
21.1.5. Triangle Lists

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST`, each consecutive set of three vertices defines a single triangle primitive, according to the equation:

$$p_i = \{v_{3i}, v_{3i+1}, v_{3i+2}\}$$

The number of primitives generated is equal to $\lfloor \text{vertexCount} / 3 \rfloor$.

The provoking vertex for p_i is v_{3i} .



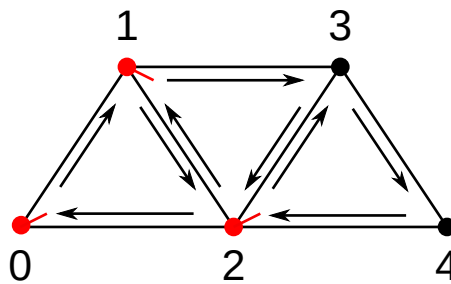
21.1.6. Triangle Strips

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP`, one triangle primitive is defined by each vertex and the two vertices that follow it, according to the equation:

$$p_i = \{v_i, v_{i+(1+i\%2)}, v_{i+(2-i\%2)}\}$$

The number of primitives generated is equal to $\max(0, \text{vertexCount} - 2)$.

The provoking vertex for p_i is v_i .



NOTE

The ordering of the vertices in each successive triangle is reversed, so that the winding order is consistent throughout the strip.

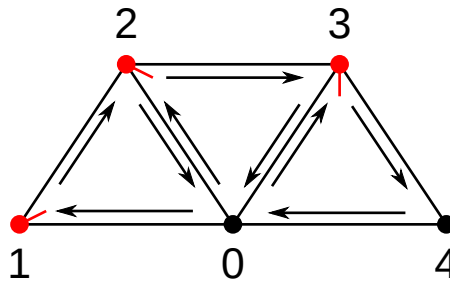
21.1.7. Triangle Fans

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN`, triangle primitives are defined around a shared common vertex, according to the equation:

$$p_i = \{v_{i+1}, v_{i+2}, v_0\}$$

The number of primitives generated is equal to $\max(0, \text{vertexCount}-2)$.

The provoking vertex for p_i is v_{i+1} .



21.1.8. Line Lists With Adjacency

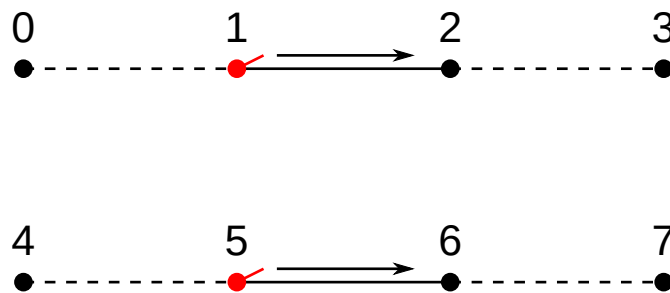
When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY`, each consecutive set of four vertices defines a single line primitive with adjacency, according to the equation:

$$p_i = \{v_{4i}, v_{4i+1}, v_{4i+2}, v_{4i+3}\}$$

A line primitive is described by the second and third vertices of the total primitive, with the remaining two vertices only accessible in a [geometry shader](#).

The number of primitives generated is equal to $\lfloor \text{vertexCount}/4 \rfloor$.

The provoking vertex for p_i is v_{4i+1} .



21.1.9. Line Strips With Adjacency

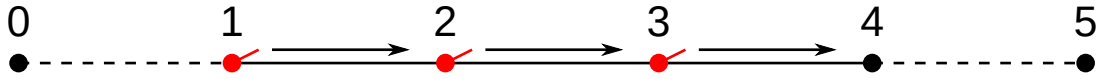
When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY`, one line primitive with adjacency is defined by each vertex and the following vertex, according to the equation:

$$p_i = \{v_i, v_{i+1}, v_{i+2}, v_{i+3}\}$$

A line primitive is described by the second and third vertices of the total primitive, with the remaining two vertices only accessible in a [geometry shader](#).

The number of primitives generated is equal to $\max(0, \text{vertexCount}-3)$.

The provoking vertex for p_i is v_{i+1} .



21.1.10. Triangle Lists With Adjacency

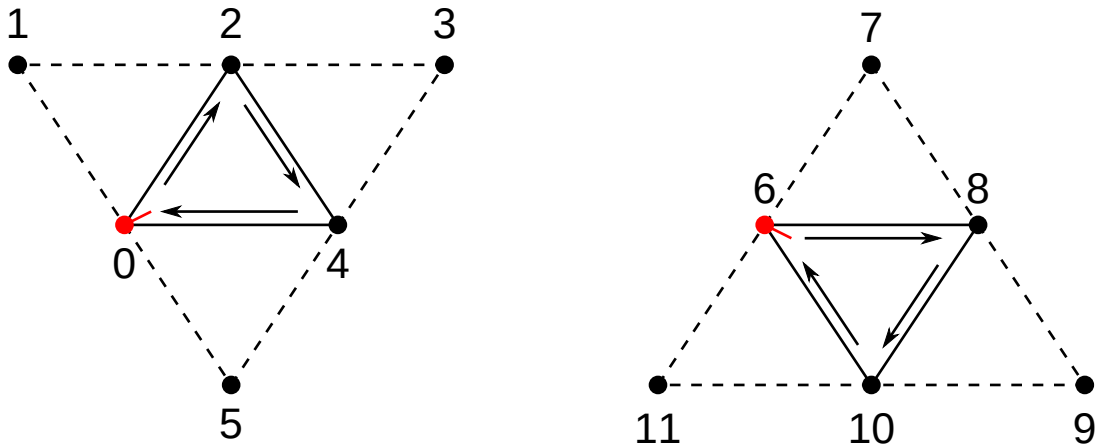
When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY`, each consecutive set of six vertices defines a single triangle primitive with adjacency, according to the equations:

$$p_i = \{v_{6i}, v_{6i+1}, v_{6i+2}, v_{6i+3}, v_{6i+4}, v_{6i+5}\}$$

A triangle primitive is described by the first, third, and fifth vertices of the total primitive, with the remaining three vertices only accessible in a [geometry shader](#).

The number of primitives generated is equal to $\lfloor \text{vertexCount} / 6 \rfloor$.

The provoking vertex for p_i is v_{6i} .



21.1.11. Triangle Strips With Adjacency

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY`, one triangle primitive with adjacency is defined by each vertex and the following 5 vertices.

The number of primitives generated, n , is equal to $\lfloor \max(0, \text{vertexCount} - 4) / 2 \rfloor$.

If $n=1$, the primitive is defined as:

$$p = \{v_0, v_1, v_2, v_3, v_4, v_5\}$$

If $n>1$, the total primitive consists of different vertices according to where it is in the strip:

$$p_i = \{v_{2i}, v_{2i+1}, v_{2i+2}, v_{2i+6}, v_{2i+4}, v_{2i+3}\} \text{ when } i=0$$

$$p_i = \{v_{2i}, v_{2i+3}, v_{2i+4}, v_{2i+6}, v_{2i+2}, v_{2i-2}\} \text{ when } i>0, i<n-1, \text{ and } i\%2=1$$

$p_i = \{v_{2i}, v_{2i-2}, v_{2i+2}, v_{2i+6}, v_{2i+4}, v_{2i+3}\}$ when $i > 0$, $i < n-1$, and $i \% 2 = 0$

$p_i = \{v_{2i}, v_{2i+3}, v_{2i+4}, v_{2i+5}, v_{2i+2}, v_{2i-2}\}$ when $i = n-1$ and $i \% 2 = 1$

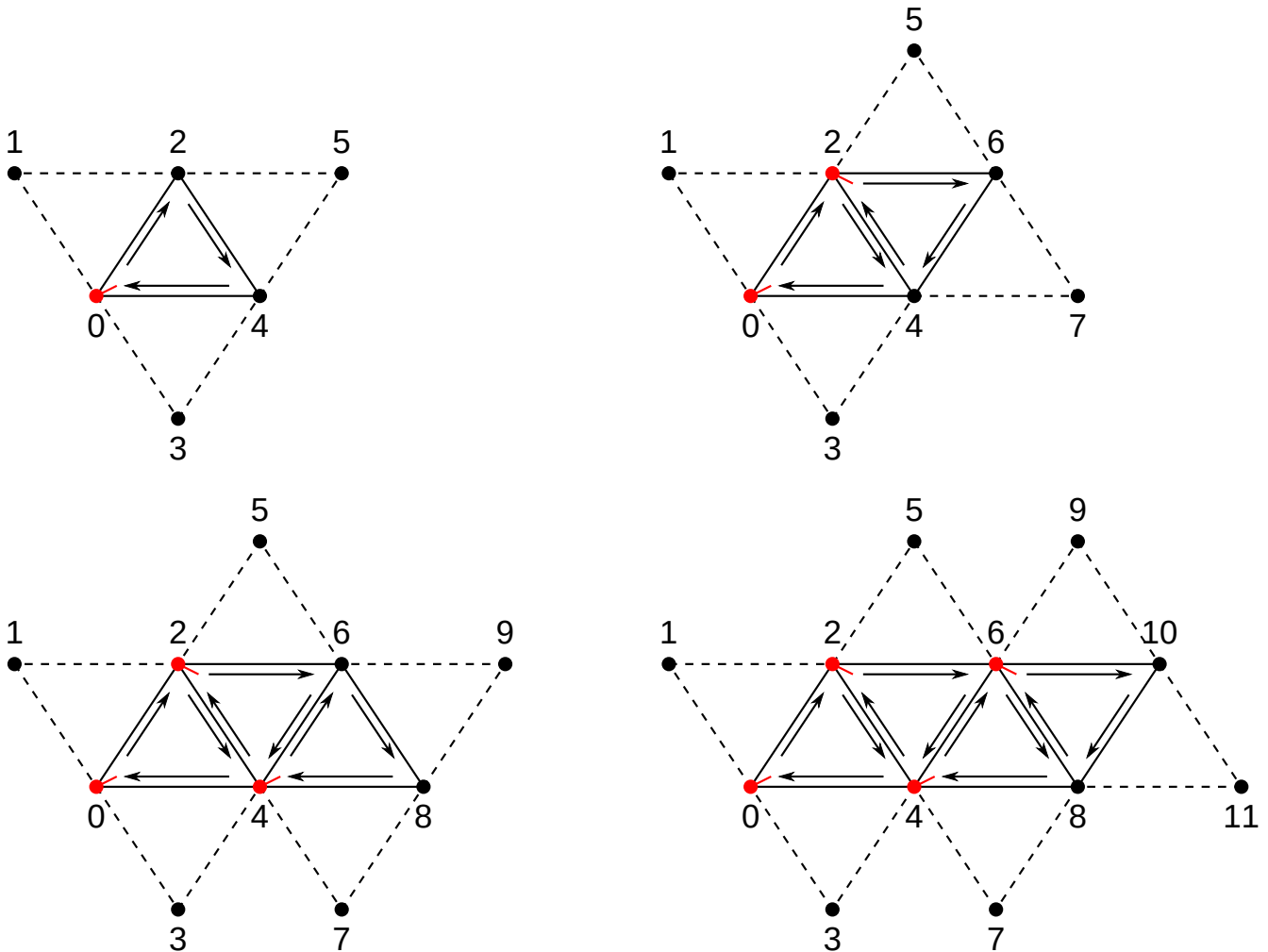
$p_i = \{v_{2i}, v_{2i-2}, v_{2i+2}, v_{2i+5}, v_{2i+4}, v_{2i+3}\}$ when $i = n-1$ and $i \% 2 = 0$

A triangle primitive is described by the first, third, and fifth vertices of the total primitive in all cases, with the remaining three vertices only accessible in a [geometry shader](#).

NOTE

The ordering of the vertices in each successive triangle is altered so that the winding order is consistent throughout the strip.

The provoking vertex for p_i is always v_{2i} .



21.1.12. Patch Lists

When the primitive topology is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, each consecutive set of m vertices defines a single patch primitive, according to the equation:

$$p_i = \{v_{mi}, v_{mi+1}, \dots, v_{mi+(m-2)}, v_{mi+(m-1)}\}$$

where m is equal to `VkPipelineTessellationStateCreateInfo::patchControlPoints`.

Patch lists are never passed to [vertex post-processing](#), and as such no provoking vertex is defined for patch primitives. The number of primitives generated is equal to $\lceil \text{vertexCount}/m \rceil$.

The vertices comprising a patch have no implied geometry, and are used as inputs to tessellation shaders and the fixed-function tessellator to generate new point, line, or triangle primitives.

21.2. Effective Primitive Topology

The effective primitive topology is the primitive topology that is consumed or produced by a given stage of the graphics pipeline. The interpretation of this topology depends on the active shader stages and other state set in the graphics pipeline.

21.2.1. Vertex Input Assembler Topology

This is the topology set when the `VK_SHADER_STAGE_VERTEX_BIT` stage is used and [Programmable Primitive Shading](#) occurs. It **can** be set with `VkPipelineInputAssemblyStateCreateInfo::topology` or `vkCmdSetPrimitiveTopologyEXT`.

21.2.2. Clip Space Topology

This is the output of the last pre-rasterization shader stage involved.

- If `VK_SHADER_STAGE_VERTEX_BIT` is the last pre-rasterization stage, the topology is not changed from what was set in the [input assembly](#).
- If `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` is the last pre-rasterization stage, the topology depends on the `OpExecutionMode` found in either [tessellation stage](#).
 - The `Isolines Execution Mode` is in line [topology class](#).
 - The `Triangles and Quads Execution Mode` is in triangle [topology class](#).
 - The `PointMode Execution Mode` is in point [topology class](#) and will take precedence over the other tessellation stage `Execution Mode`.
- If `VK_SHADER_STAGE_GEOMETRY_BIT` is the last pre-rasterization stage, the topology depends on the `OpExecutionMode` found in this shader stage.
 - The `OutputPoints Execution Mode` is in point [topology class](#).
 - The `OutputLineStrip Execution Mode` is in line [topology class](#).
 - The `OutputTriangleStrip Execution Mode` is in triangle [topology class](#).

21.2.3. Rasterization Input Topology

After clipping and viewport transformations, the [Polygon Mode](#) **can** set a topology as an input for rasterization. This **can** be different from what was used in [Clip Space Topology](#).

If `VkPipelineRasterizationStateCreateInfo::rasterizerDiscardEnable` or `vkCmdSetRasterizerDiscardEnableEXT::rasterizerDiscardEnable` is set to `VK_FALSE`, then this Rasterization Input Topology is not observed.

If the `clip space topology` is in triangle `topology class` then `VK_POLYGON_MODE_LINE` will turn it into line topology and `VK_POLYGON_MODE_POINT` will turn it into point topology.

If the `clip space topology` is in line `topology class` then `VK_POLYGON_MODE_POINT` will turn it into point topology.

21.3. Primitive Order

Primitives generated by `drawing commands` progress through the stages of the `graphics pipeline` in *primitive order*. Primitive order is initially determined in the following way:

1. Submission order determines the initial ordering
2. For indirect drawing commands, the order in which accessed instances of the `VkDrawIndirectCommand` are stored in `buffer`, from lower indirect buffer addresses to higher addresses.
3. If a drawing command includes multiple instances, the order in which instances are executed, from lower numbered instances to higher.
4. The order in which primitives are specified by a drawing command:
 - For non-indexed draws, from vertices with a lower numbered `vertexIndex` to a higher numbered `vertexIndex`.
 - For indexed draws, vertices sourced from a lower index buffer addresses to higher addresses.

Within this order implementations further sort primitives:

5. If tessellation shading is active, by an implementation-dependent order of new primitives generated by `tessellation`.
6. If geometry shading is active, by the order new primitives are generated by `geometry shading`.
7. If the `polygon mode` is not `VK_POLYGON_MODE_FILL`, by an implementation-dependent ordering of the new primitives generated within the original primitive.

Primitive order is later used to define `rasterization order`, which determines the order in which fragments output results to a framebuffer.

21.4. Programmable Primitive Shading

Once primitives are assembled, they proceed to the vertex shading stage of the pipeline. If the draw includes multiple instances, then the set of primitives is sent to the vertex shading stage multiple times, once for each instance.

It is implementation-dependent whether vertex shading occurs on vertices that are discarded as part of incomplete primitives, but if it does occur then it operates as if they were vertices in

complete primitives and such invocations **can** have side effects.

Vertex shading receives two per-vertex inputs from the primitive assembly stage - the **vertexIndex** and the **instanceIndex**. How these values are generated is defined below, with each command.

Drawing commands fall roughly into two categories:

- Non-indexed drawing commands present a sequential **vertexIndex** to the vertex shader. The sequential index is generated automatically by the device (see [Fixed-Function Vertex Processing](#) for details on both specifying the vertex attributes indexed by **vertexIndex**, as well as binding vertex buffers containing those attributes to a command buffer). These commands are:
 - **vkCmdDraw**
 - **vkCmdDrawIndirect**
 - **vkCmdDrawIndirectCount**
- Indexed drawing commands read index values from an *index buffer* and use this to compute the **vertexIndex** value for the vertex shader. These commands are:
 - **vkCmdDrawIndexed**
 - **vkCmdDrawIndexedIndirect**
 - **vkCmdDrawIndexedIndirectCount**

To bind an index buffer to a command buffer, call:

```
// Provided by VK_VERSION_1_0
void vkCmdBindIndexBuffer(
    VkCommandBuffer          commandBuffer,
    VkBuffer                 buffer,
    VkDeviceSize             offset,
    VkIndexType              indexType);
```

- **commandBuffer** is the command buffer into which the command is recorded.
- **buffer** is the buffer being bound.
- **offset** is the starting offset in bytes within **buffer** used in index buffer address calculations.
- **indexType** is a **VkIndexType** value specifying the size of the indices.

Valid Usage

- VUID-vkCmdBindIndexBuffer-offset-08782
offset must be less than the size of **buffer**
- VUID-vkCmdBindIndexBuffer-offset-08783
The sum of **offset** and the base address of the range of **VkDeviceMemory** object that is backing **buffer**, **must** be a multiple of the size of the type indicated by **indexType**
- VUID-vkCmdBindIndexBuffer-buffer-08784
buffer must have been created with the **VK_BUFFER_USAGE_INDEX_BUFFER_BIT** flag

- VUID-vkCmdBindIndexBuffer-buffer-08785
If **buffer** is non-sparse then it **must** be bound completely and contiguously to a single **VkDeviceMemory** object
- VUID-vkCmdBindIndexBuffer-indexType-08787
If **indexType** is **VK_INDEX_TYPE_UINT8**, the **indexTypeUint8** feature **must** be enabled
- VUID-vkCmdBindIndexBuffer-None-09493
buffer **must** not be **VK_NULL_HANDLE**

Valid Usage (Implicit)

- VUID-vkCmdBindIndexBuffer-commandBuffer-parameter
commandBuffer **must** be a valid **VkCommandBuffer** handle
- VUID-vkCmdBindIndexBuffer-buffer-parameter
If **buffer** is not **VK_NULL_HANDLE**, **buffer** **must** be a valid **VkBuffer** handle
- VUID-vkCmdBindIndexBuffer-indexType-parameter
indexType **must** be a valid **VkIndexType** value
- VUID-vkCmdBindIndexBuffer-commandBuffer-recording
commandBuffer **must** be in the **recording state**
- VUID-vkCmdBindIndexBuffer-commandBuffer-cmdpool
The **VkCommandPool** that **commandBuffer** was allocated from **must** support **VK_QUEUE_GRAPHICS_BIT** operations
- VUID-vkCmdBindIndexBuffer-commonparent
Both of **buffer**, and **commandBuffer** that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same **VkDevice**

Host Synchronization

- Host access to **commandBuffer** **must** be externally synchronized
- Host access to the **VkCommandPool** that **commandBuffer** was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

Possible values of **vkCmdBindIndexBuffer::indexType**, specifying the size of indices, are:


```
// Provided by VK_VERSION_1_0
typedef enum VkIndexType {
    VK_INDEX_TYPE_UINT16 = 0,
    VK_INDEX_TYPE_UINT32 = 1,
    VK_INDEX_TYPE_UINT8 = 1000265000,
    // Provided by VK_EXT_index_type_uint8
    VK_INDEX_TYPE_UINT8_EXT = VK_INDEX_TYPE_UINT8,
    // Provided by VK_KHR_index_type_uint8
    VK_INDEX_TYPE_UINT8_KHR = VK_INDEX_TYPE_UINT8,
} VkIndexType;
```

- `VK_INDEX_TYPE_UINT16` specifies that indices are 16-bit unsigned integer values.
- `VK_INDEX_TYPE_UINT32` specifies that indices are 32-bit unsigned integer values.
- `VK_INDEX_TYPE_UINT8` specifies that indices are 8-bit unsigned integer values.

The parameters for each drawing command are specified directly in the command or read from buffer memory, depending on the command. Drawing commands that source their parameters from buffer memory are known as *indirect* drawing commands.

All drawing commands interact with the `robustBufferAccess` feature.

To record a non-indexed draw, call:

```
// Provided by VK_VERSION_1_0
void vkCmdDraw(
    VkCommandBuffer          commandBuffer,
    uint32_t                 vertexCount,
    uint32_t                 instanceCount,
    uint32_t                 firstVertex,
    uint32_t                 firstInstance);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `vertexCount` is the number of vertices to draw.
- `instanceCount` is the number of instances to draw.
- `firstVertex` is the index of the first vertex to draw.
- `firstInstance` is the instance ID of the first instance to draw.

When the command is executed, primitives are assembled using the current primitive topology and `vertexCount` consecutive vertex indices with the first `vertexIndex` value equal to `firstVertex`. The primitives are drawn `instanceCount` times with `instanceIndex` starting with `firstInstance` and increasing sequentially for each instance. The assembled primitives execute the bound graphics pipeline.

Valid Usage

- VUID-vkCmdDraw-magFilter-04553

If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDraw-magFilter-09598

If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`

- VUID-vkCmdDraw-mipmapMode-04770

If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDraw-mipmapMode-09599

If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`

- VUID-vkCmdDraw-unnormalizedCoordinates-09635

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `levelCount` and `layerCount` **must** be 1

- VUID-vkCmdDraw-None-08609

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `viewType` **must** be `VK_IMAGE_VIEW_TYPE_1D` or `VK_IMAGE_VIEW_TYPE_2D`

- VUID-vkCmdDraw-None-08610

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name

- VUID-vkCmdDraw-None-08611

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values

- VUID-vkCmdDraw-aspectMask-06478
If a [VkImageView](#) is sampled with [depth comparison](#), the image view **must** have been created with an [aspectMask](#) that contains [VK_IMAGE_ASPECT_DEPTH_BIT](#)
- VUID-vkCmdDraw-None-02691
If a [VkImageView](#) is accessed using atomic operations as a result of this command, then the image view's [format](#) [features](#) **must** contain [VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT](#)
- VUID-vkCmdDraw-None-07888
If a [VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER](#) descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's [format](#) [features](#) **must** contain [VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT](#)
- VUID-vkCmdDraw-None-02692
If a [VkImageView](#) is sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command, then the image view's [format](#) [features](#) **must** contain [VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT](#)
- VUID-vkCmdDraw-None-02693
If the [VK_EXT_filter_cubic](#) extension is not enabled and any [VkImageView](#) is sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command, it **must** not have a [VkImageViewType](#) of [VK_IMAGE_VIEW_TYPE_3D](#), [VK_IMAGE_VIEW_TYPE_CUBE](#), or [VK_IMAGE_VIEW_TYPE_CUBE_ARRAY](#)
- VUID-vkCmdDraw-filterCubic-02694
Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)
- VUID-vkCmdDraw-filterCubicMinmax-02695
Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) with a reduction mode of either [VK_SAMPLER_REDUCTION_MODE_MIN](#) or [VK_SAMPLER_REDUCTION_MODE_MAX](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering together with minmax filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)
- VUID-vkCmdDraw-None-08600
For each set *n* that is statically used by a [bound shader](#), a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for set *n*, with the [VkPipelineLayout](#) used to create the current [VkPipeline](#) , as described in [Pipeline Layout Compatibility](#)
- VUID-vkCmdDraw-None-08601
For each push constant that is statically used by a [bound shader](#), a push constant value **must** have been set for the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for push constants, with the [VkPipelineLayout](#) used to create the current [VkPipeline](#) , as described in [Pipeline Layout Compatibility](#)
- VUID-vkCmdDraw-None-10068
For each array of resources that is used by a [bound shader](#), the indices used to access members of the array **must** be less than the descriptor count for the identified binding in

the descriptor sets used by this command

- VUID-vkCmdDraw-None-08114

Descriptors in each bound descriptor set, specified via [vkCmdBindDescriptorSets](#), **must** be valid as described by [descriptor validity](#) if they are statically used by a [bound shader](#)

- VUID-vkCmdDraw-imageLayout-00344

If an image descriptor is accessed by a shader, the [VkImageLayout](#) **must** match the subresource accessible from the [VkImageView](#) as defined by the [image layout matching rules](#)

- VUID-vkCmdDraw-None-08606

A valid pipeline **must** be bound to the pipeline bind point used by this command

- VUID-vkCmdDraw-None-08608

There **must** not have been any calls to dynamic state setting commands for any state specified statically in the [VkPipeline](#) object bound to the pipeline bind point used by this command, since that pipeline was bound

- VUID-vkCmdDraw-uniformBuffers-06935

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a uniform buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDraw-storageBuffers-06936

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a storage buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDraw-commandBuffer-02707

If [commandBuffer](#) is an unprotected command buffer and [protectedNoFault](#) is not supported, any resource accessed by [bound shaders](#) **must** not be a protected resource

- VUID-vkCmdDraw-viewType-07752

If a [VkImageView](#) is accessed as a result of this command, then the image view's [viewType](#) **must** match the [Dim](#) operand of the [OpTypeImage](#) as described in [Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types](#)

- VUID-vkCmdDraw-format-07753

If a [VkImageView](#) or [VkBufferView](#) is accessed as a result of this command, then the [numeric type](#) of the view's [format](#) and the [Sampled Type](#) operand of the [OpTypeImage](#) **must** match

- VUID-vkCmdDraw-OpImageWrite-08795

If a [VkImageView](#) is accessed using [OpImageWrite](#) as a result of this command, then the [Type](#) of the [Texel](#) operand of that instruction **must** have at least as many components as the image view's format

- VUID-vkCmdDraw-OpImageWrite-04469

If a [VkBufferView](#) is accessed using [OpImageWrite](#) as a result of this command, then the [Type](#) of the [Texel](#) operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDraw-SampledType-04470
If a `VkImageView` with a `VkFormat` that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDraw-SampledType-04471
If a `VkImageView` with a `VkFormat` that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDraw-SampledType-04472
If a `VkBufferView` with a `VkFormat` that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64
- VUID-vkCmdDraw-SampledType-04473
If a `VkBufferView` with a `VkFormat` that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32
- VUID-vkCmdDraw-sparseImageInt64Atomics-04474
If the `sparseImageInt64Atomics` feature is not enabled, `VkImage` objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDraw-sparseImageInt64Atomics-04475
If the `sparseImageInt64Atomics` feature is not enabled, `VkBuffer` objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDraw-None-07288
Any shader invocation executed by this command **must** `terminate`
- VUID-vkCmdDraw-None-09600
If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, all image subresources identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDraw-renderPass-02684
The current render pass **must** be `compatible` with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDraw-subpass-02685
The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDraw-None-07748
If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDraw-OpTypeImage-07468

If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass

- VUID-vkCmdDraw-None-07469

Input attachment views accessed in a subpass **must** be created with the same `VkFormat` as the corresponding subpass definition, and be created with a `VkImageView` that is compatible with the attachment referenced by the subpass' `pInputAttachments` [`InputAttachmentIndex`] in the bound `VkFramebuffer` as specified by [Fragment Input Attachment Compatibility](#)

- VUID-vkCmdDraw-None-06537

Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command

- VUID-vkCmdDraw-None-10795

If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDraw-None-10796

If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDraw-None-10797

If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDraw-None-06539

If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment

- VUID-vkCmdDraw-None-06886

If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, `depth writes` **must** be disabled

- VUID-vkCmdDraw-None-06887

If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back `writeMask` are not zero, and stencil test is enabled, `all stencil ops` **must** be `VK_STENCIL_OP_KEEP`

- VUID-vkCmdDraw-None-07831

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled then `vkCmdSetViewport` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07832

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled then `vkCmdSetScissor` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-08617

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `effective rasterization input topology` is in line topology class, then `vkCmdSetLineWidth` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07834

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthBiasEnable` is `VK_TRUE`, then `vkCmdSetDepthBias` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07835

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_BLEND_CONSTANTS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and an active color attachment `current value` of `blendEnable` is `VK_TRUE` with a blend equations where any `VkBlendFactor` member is `VK_BLEND_FACTOR_CONSTANT_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR`, `VK_BLEND_FACTOR_CONSTANT_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA`, then `vkCmdSetBlendConstants` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07836

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BOUNDS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthBoundsTestEnable` is `VK_TRUE`, then `vkCmdSetDepthBounds` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07837

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilCompareMask` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07838

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilWriteMask` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07839

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_REFERENCE` dynamic state enabled, the `current value` of and

`rasterizerDiscardEnable` is `VK_FALSE`, the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilReference` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-maxMultiviewInstanceIndex-02688

If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to `VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex`

- VUID-vkCmdDraw-sampleLocationsEnable-02689

If the bound graphics pipeline was created with `VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable` set to `VK_TRUE`, then the active depth attachment **must** have been created with the `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` bit set

- VUID-vkCmdDraw-None-06666

If the `VK_EXT_sample_locations` extension is enabled, a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `sampleLocationsEnable` is `VK_TRUE`, then `vkCmdSetSampleLocationsEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07840

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_CULL_MODE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetCullModeEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07841

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_FRONT_FACE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetFrontFaceEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07843

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, `vkCmdSetDepthTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07844

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthTestEnable` is `VK_TRUE`, then `vkCmdSetDepthWriteEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07845

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthTestEnable` is `VK_TRUE`,

then `vkCmdSetDepthCompareOpEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07846

If the `depthBounds` feature is enabled, a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetDepthBoundsTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07847

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetStencilTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-07848

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_OP` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilOpEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-viewportCount-03417

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, then `vkCmdSetViewportWithCountEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-scissorCount-03418

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` dynamic state enabled, then `vkCmdSetScissorWithCountEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing

- VUID-vkCmdDraw-viewportCount-03419

If a graphics pipeline is bound which was created with both the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic states enabled, then the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** match the `scissorCount` parameter of `vkCmdSetScissorWithCountEXT`

- VUID-vkCmdDraw-None-04876

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` dynamic state enabled, then `vkCmdSetRasterizerDiscardEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-04877

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetDepthBiasEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this

drawing command

- VUID-vkCmdDraw-logicOp-04878

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LOGIC_OP_EXT` dynamic state enabled, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, and the **current value** of `logicOpEnable` is `VK_TRUE`, then `vkCmdSetLogicOpEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-primitiveFragmentShadingRateWithMultipleViewports-04552

If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the `PrimitiveShadingRateKHR` built-in, then `vkCmdSetViewportWithCountEXT` **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be 1

- VUID-vkCmdDraw-blendEnable-04727

If a graphics pipeline is bound which was created with `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic state enabled, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, then for each color attachment, if the corresponding image view's **format features** do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the corresponding **current value** of `blendEnable` **must** be `VK_FALSE`

- VUID-vkCmdDraw-None-08644

If a graphics pipeline is bound, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`,

then the **current value** of `rasterizationSamples` **must** be the same as the current color and/or depth/stencil attachments

- VUID-vkCmdDraw-maxFragmentDualSrcAttachments-09239

If **blending** is enabled for any attachment where either the source or destination blend factors for that attachment **use the secondary color input**, the maximum value of `Location` for any output attachment **statically used** in the `Fragment Execution Model` executed by this command **must** be less than `maxFragmentDualSrcAttachments`

- VUID-vkCmdDraw-commandBuffer-02712

If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, any resource written to by the `VkPipeline` object bound to the pipeline bind point used by this command **must** not be an unprotected resource

- VUID-vkCmdDraw-commandBuffer-02713

If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, pipeline stages other than the framebuffer-space and compute stages in the `VkPipeline` object bound to the pipeline bind point used by this command **must** not write to any resource

- VUID-vkCmdDraw-None-04007

All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or `VK_NULL_HANDLE` buffers bound

- VUID-vkCmdDraw-None-04008
If the `nullDescriptor` feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be `VK_NULL_HANDLE`
- VUID-vkCmdDraw-None-02721
If the `robustBufferAccess` feature is not enabled, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in [Vertex Input Description](#)
- VUID-vkCmdDraw-format-10389
For each vertex attribute accessed by this command, if its `VkVertexInputAttributeDescription::format` or `VkVertexInputAttributeDescription2EXT::format` is a `packed format`, the value of `attribAddress`, calculated as described in [Vertex Input Calculation](#), **must** be a multiple of the `size of the format`
- VUID-vkCmdDraw-format-10390
For each vertex attribute accessed by this command, if its `VkVertexInputAttributeDescription::format` or `VkVertexInputAttributeDescription2EXT::format` is not a `packed format`, the value of `attribAddress`, calculated as described in [Vertex Input Calculation](#), **must** be a multiple of the `component size of the format`
- VUID-vkCmdDraw-None-07842
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then `vkCmdSetPrimitiveTopologyEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command
- VUID-vkCmdDraw-dynamicPrimitiveTopologyUnrestricted-07500
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then the `primitiveTopology` parameter of `vkCmdSetPrimitiveTopologyEXT` **must** be of the same `topology class` as the pipeline `VkPipelineInputAssemblyStateCreateInfo::topology` state
- VUID-vkCmdDraw-primitiveTopology-10286
If a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage is bound, then the `current value` of `primitiveTopology` **must** be `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST` prior to this drawing command
- VUID-vkCmdDraw-primitiveTopology-10747
If `vkCmdSetPrimitiveTopologyEXT` set `primitiveTopology` to `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST` prior to this drawing command, then a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage **must** be bound
- VUID-vkCmdDraw-primitiveTopology-10748
If `vkCmdSetPrimitiveTopologyEXT` set `primitiveTopology` to `VK_PRIMITIVE_TOPOLOGY_POINT_LIST` prior to this drawing command, both a `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` and `VK_SHADER_STAGE_GEOMETRY_BIT` stage are not bound, then the `Vertex Execution Model` **must** have a `PointSize` decorated variable that is statically written to
- VUID-vkCmdDraw-pStrides-04913
If the bound graphics pipeline was created with the

`VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE` dynamic state enabled, but without the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, then `vkCmdBindVertexBuffers2EXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this draw command, and the `pStrides` parameter of `vkCmdBindVertexBuffers2EXT` **must** not be `NULL`

- VUID-vkCmdDraw-None-04914

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then `vkCmdSetVertexInputEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this draw command

- VUID-vkCmdDraw-Input-07939

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then all variables with the `Input` storage class decorated with `Location` in the `Vertex Execution Model OpEntryPoint` **must** contain a location in `VkVertexInputAttributeDescription2EXT::location`

- VUID-vkCmdDraw-Input-08734

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then the numeric type associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be the same as `VkVertexInputAttributeDescription2EXT::format`

- VUID-vkCmdDraw-format-08936

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and `VkVertexInputAttributeDescription2EXT::format` has a 64-bit component, then the scalar width associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be 64-bit

- VUID-vkCmdDraw-format-08937

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and the scalar width associated with a `Location` decorated `Input` variable in the `Vertex Execution Model OpEntryPoint` is 64-bit, then the corresponding `VkVertexInputAttributeDescription2EXT::format` **must** have a 64-bit component

- VUID-vkCmdDraw-None-09203

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and `VkVertexInputAttributeDescription2EXT::format` has a 64-bit component, then all `Input` variables at the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** not use components that are not present in the format

- VUID-vkCmdDraw-None-04875

If the bound graphics pipeline state was created with both a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage and the `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT` dynamic state enabled, and the `current value` of `primitiveTopology` is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, then `vkCmdSetPatchControlPointsEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-04879

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled then `vkCmdSetPrimitiveRestartEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDraw-None-09637

If the `input assembly` is `VK_PRIMITIVE_TOPOLOGY_POINT_LIST`, `VK_PRIMITIVE_TOPOLOGY_LINE_LIST`, `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST`, `VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY`, or `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY`, the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled, then `vkCmdSetPrimitiveRestartEnableEXT` **must** be `VK_FALSE`

- VUID-vkCmdDraw-None-10909

If the `input assembly` is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled then `vkCmdSetPrimitiveRestartEnableEXT` **must** be `VK_FALSE`

- VUID-vkCmdDraw-pNext-09461

If the bound graphics pipeline state was created with `VkPipelineVertexInputDivisorStateCreateInfo` in the `pNext` chain of `VkGraphicsPipelineCreateInfo::pVertexInputState`, any member of `VkPipelineVertexInputDivisorStateCreateInfo::pVertexBindingDivisors` has a value other than `1` in `divisor`, and `VkPhysicalDeviceVertexAttributeDivisorProperties::supportsNonZeroFirstInstance` is `VK_FALSE`, then `firstInstance` **must** be `0`

- VUID-vkCmdDraw-None-09462

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, any member of the `pVertexBindingDescriptions` parameter to the `vkCmdSetVertexInputEXT` call that sets this dynamic state has a value other than `1` in `divisor`, and `VkPhysicalDeviceVertexAttributeDivisorProperties::supportsNonZeroFirstInstance` is `VK_FALSE`, then `firstInstance` **must** be `0`

Valid Usage (Implicit)

- VUID-vkCmdDraw-commandBuffer-parameter

`commandBuffer` **must** be a valid `VkCommandBuffer` handle

- VUID-vkCmdDraw-commandBuffer-recording

`commandBuffer` **must** be in the `recording` state

- VUID-vkCmdDraw-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

- VUID-vkCmdDraw-renderpass

This command **must** only be called inside of a render pass instance

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Inside	VK_QUEUE_GRAPHICS_BIT	Action

To record an indexed draw, call:

```
// Provided by VK_VERSION_1_0
void vkCmdDrawIndexed(
    VkCommandBuffer          commandBuffer,
    uint32_t                 indexCount,
    uint32_t                 instanceCount,
    uint32_t                 firstIndex,
    int32_t                  vertexOffset,
    uint32_t                 firstInstance);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `indexCount` is the number of vertices to draw.
- `instanceCount` is the number of instances to draw.
- `firstIndex` is the base index within the index buffer.
- `vertexOffset` is the value added to the vertex index before indexing into the vertex buffer.
- `firstInstance` is the instance ID of the first instance to draw.

When the command is executed, primitives are assembled using the current primitive topology and `indexCount` vertices whose indices are retrieved from the index buffer. The index buffer is treated as an array of tightly packed unsigned integers of size defined by the `vkCmdBindIndexBuffer::indexType` parameter with which the buffer was bound.

The first vertex index is at an offset of $\text{firstIndex} \times \text{indexSize} + \text{offset}$ within the bound index buffer, where `offset` is the offset specified by `vkCmdBindIndexBuffer` and `indexSize` is the byte size of the type specified by `indexType`. Subsequent index values are retrieved from consecutive locations in the index buffer. Indices are first compared to the primitive restart value, then zero extended to 32 bits (if the `indexType` is `VK_INDEX_TYPE_UINT8` or `VK_INDEX_TYPE_UINT16`) and have `vertexOffset` added to them, before being supplied as the `vertexIndex` value.

The primitives are drawn `instanceCount` times with `instanceIndex` starting with `firstInstance` and increasing sequentially for each instance. The assembled primitives execute the bound graphics pipeline.

Valid Usage

- VUID-vkCmdDrawIndexed-magFilter-04553

If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDrawIndexed-magFilter-09598

If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`

- VUID-vkCmdDrawIndexed-mipmapMode-04770

If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDrawIndexed-mipmapMode-09599

If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`

- VUID-vkCmdDrawIndexed-unnormalizedCoordinates-09635

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `levelCount` and `layerCount` **must** be 1

- VUID-vkCmdDrawIndexed-None-08609

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `viewType` **must** be `VK_IMAGE_VIEW_TYPE_1D` or `VK_IMAGE_VIEW_TYPE_2D`

- VUID-vkCmdDrawIndexed-None-08610

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name

- VUID-vkCmdDrawIndexed-None-08611

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a

[VkImageView](#) as a result of this command, then the sampler **must** not be used with any of the SPIR-V [OpImageSample*](#) or [OpImageSparseSample*](#) instructions that includes a LOD bias or any offset values

- VUID-vkCmdDrawIndexed-aspectMask-06478

If a [VkImageView](#) is sampled with [depth comparison](#), the image view **must** have been created with an [aspectMask](#) that contains [VK_IMAGE_ASPECT_DEPTH_BIT](#)

- VUID-vkCmdDrawIndexed-None-02691

If a [VkImageView](#) is accessed using atomic operations as a result of this command, then the image view's [format](#) [features](#) **must** contain [VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT](#)

- VUID-vkCmdDrawIndexed-None-07888

If a [VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER](#) descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's [format](#) [features](#) **must** contain [VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT](#)

- VUID-vkCmdDrawIndexed-None-02692

If a [VkImageView](#) is sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command, then the image view's [format](#) [features](#) **must** contain [VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT](#)

- VUID-vkCmdDrawIndexed-None-02693

If the [VK_EXT_filter_cubic](#) extension is not enabled and any [VkImageView](#) is sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command, it **must** not have a [VkImageViewType](#) of [VK_IMAGE_VIEW_TYPE_3D](#), [VK_IMAGE_VIEW_TYPE_CUBE](#), or [VK_IMAGE_VIEW_TYPE_CUBE_ARRAY](#)

- VUID-vkCmdDrawIndexed-filterCubic-02694

Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)

- VUID-vkCmdDrawIndexed-filterCubicMinmax-02695

Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) with a reduction mode of either [VK_SAMPLER_REDUCTION_MODE_MIN](#) or [VK_SAMPLER_REDUCTION_MODE_MAX](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering together with minmax filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)

- VUID-vkCmdDrawIndexed-None-08600

For each set n that is statically used by a [bound shader](#), a descriptor set **must** have been bound to n at the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for set n , with the [VkPipelineLayout](#) used to create the current [VkPipeline](#), as described in [Pipeline Layout Compatibility](#)

- VUID-vkCmdDrawIndexed-None-08601

For each push constant that is statically used by a [bound shader](#), a push constant value **must** have been set for the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for push constants, with the [VkPipelineLayout](#) used to create the current [VkPipeline](#), as described in [Pipeline Layout Compatibility](#)

- VUID-vkCmdDrawIndexed-None-10068
For each array of resources that is used by a [bound shader](#), the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDrawIndexed-None-08114
Descriptors in each bound descriptor set, specified via [vkCmdBindDescriptorSets](#), **must** be valid as described by [descriptor validity](#) if they are statically used by a [bound shader](#)
- VUID-vkCmdDrawIndexed-imageLayout-00344
If an image descriptor is accessed by a shader, the [VkImageLayout](#) **must** match the subresource accessible from the [VkImageView](#) as defined by the [image layout matching rules](#)
- VUID-vkCmdDrawIndexed-None-08606
A valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDrawIndexed-None-08608
There **must** not have been any calls to dynamic state setting commands for any state specified statically in the [VkPipeline](#) object bound to the pipeline bind point used by this command, since that pipeline was bound
- VUID-vkCmdDrawIndexed-uniformBuffers-06935
If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a uniform buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndexed-storageBuffers-06936
If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a storage buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point
- VUID-vkCmdDrawIndexed-commandBuffer-02707
If [commandBuffer](#) is an unprotected command buffer and [protectedNoFault](#) is not supported, any resource accessed by [bound shaders](#) **must** not be a protected resource
- VUID-vkCmdDrawIndexed-viewType-07752
If a [VkImageView](#) is accessed as a result of this command, then the image view's [viewType](#) **must** match the [Dim](#) operand of the [OpTypeImage](#) as described in [Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types](#)
- VUID-vkCmdDrawIndexed-format-07753
If a [VkImageView](#) or [VkBufferView](#) is accessed as a result of this command, then the [numeric type](#) of the view's [format](#) and the [Sampled Type](#) operand of the [OpTypeImage](#) **must** match
- VUID-vkCmdDrawIndexed-OpImageWrite-08795
If a [VkImageView](#) is accessed using [OpImageWrite](#) as a result of this command, then the [Type](#) of the [Texel](#) operand of that instruction **must** have at least as many components as the image view's format
- VUID-vkCmdDrawIndexed-OpImageWrite-04469

If a `VkBufferView` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDrawIndexed-SampledType-04470

If a `VkImageView` with a `VkFormat` that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64

- VUID-vkCmdDrawIndexed-SampledType-04471

If a `VkImageView` with a `VkFormat` that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32

- VUID-vkCmdDrawIndexed-SampledType-04472

If a `VkBufferView` with a `VkFormat` that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64

- VUID-vkCmdDrawIndexed-SampledType-04473

If a `VkBufferView` with a `VkFormat` that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32

- VUID-vkCmdDrawIndexed-sparseImageInt64Atomics-04474

If the `sparseImageInt64Atomics` feature is not enabled, `VkImage` objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command

- VUID-vkCmdDrawIndexed-sparseImageInt64Atomics-04475

If the `sparseImageInt64Atomics` feature is not enabled, `VkBuffer` objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command

- VUID-vkCmdDrawIndexed-None-07288

Any shader invocation executed by this command **must** `terminate`

- VUID-vkCmdDrawIndexed-None-09600

If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, all image subresources identified by that descriptor **must** be in the image layout identified when the descriptor was written

- VUID-vkCmdDrawIndexed-renderPass-02684

The current render pass **must** be `compatible` with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`

- VUID-vkCmdDrawIndexed-subpass-02685

The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`

- VUID-vkCmdDrawIndexed-None-07748

If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set

- VUID-vkCmdDrawIndexed-OpTypeImage-07468

If any shader executed by this pipeline accesses an **OpTypeImage** variable with a **Dim** operand of **SubpassData**, it **must** be decorated with an **InputAttachmentIndex** that corresponds to a valid input attachment in the current subpass

- VUID-vkCmdDrawIndexed-None-07469

Input attachment views accessed in a subpass **must** be created with the same **VkFormat** as the corresponding subpass definition, and be created with a **VkImageView** that is compatible with the attachment referenced by the subpass' **pInputAttachments** [**InputAttachmentIndex**] in the bound **VkFramebuffer** as specified by **Fragment Input Attachment Compatibility**

- VUID-vkCmdDrawIndexed-None-06537

Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexed-None-10795

If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexed-None-10796

If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexed-None-10797

If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexed-None-06539

If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment

- VUID-vkCmdDrawIndexed-None-06886

If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, **depth writes must** be disabled

- VUID-vkCmdDrawIndexed-None-06887

If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back **writeMask** are not zero, and stencil test is enabled, **all stencil ops must** be **VK_STENCIL_OP_KEEP**

- VUID-vkCmdDrawIndexed-None-07831

If the bound graphics pipeline state was created with the **VK_DYNAMIC_STATE_VIEWPORT**

dynamic state enabled then `vkCmdSetViewport` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07832

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled then `vkCmdSetScissor` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-08617

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `effective rasterization input topology` is in line topology class, then `vkCmdSetLineWidth` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07834

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthBiasEnable` is `VK_TRUE`, then `vkCmdSetDepthBias` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07835

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_BLEND_CONSTANTS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and an active color attachment `current value` of `blendEnable` is `VK_TRUE` with a blend equations where any `VkBlendFactor` member is `VK_BLEND_FACTOR_CONSTANT_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR`, `VK_BLEND_FACTOR_CONSTANT_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA`, then `vkCmdSetBlendConstants` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07836

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BOUNDS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthBoundsTestEnable` is `VK_TRUE`, then `vkCmdSetDepthBounds` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07837

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilCompareMask` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07838

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilWriteMask` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07839

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_REFERENCE` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilReference` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-maxMultiviewInstanceIndex-02688

If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to `VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex`

- VUID-vkCmdDrawIndexed-sampleLocationsEnable-02689

If the bound graphics pipeline was created with `VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable` set to `VK_TRUE`, then the active depth attachment **must** have been created with the `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` bit set

- VUID-vkCmdDrawIndexed-None-06666

If the `VK_EXT_sample_locations` extension is enabled, a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `sampleLocationsEnable` is `VK_TRUE`, then `vkCmdSetSampleLocationsEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07840

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_CULL_MODE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetCullModeEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07841

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_FRONT_FACE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetFrontFaceEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07843

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, `vkCmdSetDepthTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07844

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthTestEnable` is `VK_TRUE`, then `vkCmdSetDepthWriteEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07845

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP` dynamic state enabled, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, and the **current value** of `depthTestEnable` is `VK_TRUE`, then `vkCmdSetDepthCompareOpEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07846

If the `depthBounds` feature is enabled, a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE` dynamic state enabled, and the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetDepthBoundsTestEnableEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07847

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE` dynamic state enabled, and the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetStencilTestEnableEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-07848

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_OP` dynamic state enabled, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, the **current value** of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilOpEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-viewportCount-03417

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, then `vkCmdSetViewportWithCountEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-scissorCount-03418

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` dynamic state enabled, then `vkCmdSetScissorWithCountEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing

- VUID-vkCmdDrawIndexed-viewportCount-03419

If a graphics pipeline is bound which was created with both the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic states enabled, then the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** match the `scissorCount` parameter of `vkCmdSetScissorWithCountEXT`

- VUID-vkCmdDrawIndexed-None-04876

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` dynamic state enabled, then `vkCmdSetRasterizerDiscardEnableEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-04877

If a graphics pipeline is bound which was created with the

`VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetDepthBiasEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-logicOp-04878

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LOGIC_OP_EXT` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `logicOpEnable` is `VK_TRUE`, then `vkCmdSetLogicOpEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-primitiveFragmentShadingRateWithMultipleViewports-04552

If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the `PrimitiveShadingRateKHR` built-in, then `vkCmdSetViewportWithCountEXT` **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be 1

- VUID-vkCmdDrawIndexed-blendEnable-04727

If a graphics pipeline is bound which was created with `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then for each color attachment, if the corresponding image view's `format` `features` do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the corresponding `current value` of `blendEnable` **must** be `VK_FALSE`

- VUID-vkCmdDrawIndexed-None-08644

If a graphics pipeline is bound, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`,

then the `current value` of `rasterizationSamples` **must** be the same as the current color and/or depth/stencil attachments

- VUID-vkCmdDrawIndexed-maxFragmentDualSrcAttachments-09239

If `blending` is enabled for any attachment where either the source or destination blend factors for that attachment `use the secondary color input`, the maximum value of `Location` for any output attachment `statically used` in the `Fragment Execution Model` executed by this command **must** be less than `maxFragmentDualSrcAttachments`

- VUID-vkCmdDrawIndexed-commandBuffer-02712

If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, any resource written to by the `VkPipeline` object bound to the pipeline bind point used by this command **must** not be an unprotected resource

- VUID-vkCmdDrawIndexed-commandBuffer-02713

If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, pipeline stages other than the framebuffer-space and compute stages in the `VkPipeline` object bound to the pipeline bind point used by this command **must** not write to any resource

- VUID-vkCmdDrawIndexed-None-04007

All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or [VK_NULL_HANDLE](#) buffers bound

- VUID-vkCmdDrawIndexed-None-04008

If the [nullDescriptor](#) feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be [VK_NULL_HANDLE](#)

- VUID-vkCmdDrawIndexed-None-02721

If the [robustBufferAccess](#) feature is not enabled, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in [Vertex Input Description](#)

- VUID-vkCmdDrawIndexed-format-10389

For each vertex attribute accessed by this command, if its [VkVertexInputAttributeDescription::format](#) or [VkVertexInputAttributeDescription2EXT::format](#) is a [packed format](#), the value of [attribAddress](#), calculated as described in [Vertex Input Calculation](#), **must** be a multiple of the [size of the format](#)

- VUID-vkCmdDrawIndexed-format-10390

For each vertex attribute accessed by this command, if its [VkVertexInputAttributeDescription::format](#) or [VkVertexInputAttributeDescription2EXT::format](#) is not a [packed format](#), the value of [attribAddress](#), calculated as described in [Vertex Input Calculation](#), **must** be a multiple of the [component size of the format](#)

- VUID-vkCmdDrawIndexed-None-07842

If the bound graphics pipeline state was created with the [VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY](#) dynamic state enabled then [vkCmdSetPrimitiveTopologyEXT](#) **must** have been called and not subsequently [invalidated](#) in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-dynamicPrimitiveTopologyUnrestricted-07500

If the bound graphics pipeline state was created with the [VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY](#) dynamic state enabled then the [primitiveTopology](#) parameter of [vkCmdSetPrimitiveTopologyEXT](#) **must** be of the same [topology class](#) as the pipeline [VkPipelineInputAssemblyStateCreateInfo::topology](#) state

- VUID-vkCmdDrawIndexed-primitiveTopology-10286

If a [VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT](#) stage is bound, then the [current value](#) of [primitiveTopology](#) **must** be [VK_PRIMITIVE_TOPOLOGY_PATCH_LIST](#) prior to this drawing command

- VUID-vkCmdDrawIndexed-primitiveTopology-10747

If [vkCmdSetPrimitiveTopologyEXT](#) set [primitiveTopology](#) to [VK_PRIMITIVE_TOPOLOGY_PATCH_LIST](#) prior to this drawing command, then a [VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT](#) stage **must** be bound

- VUID-vkCmdDrawIndexed-primitiveTopology-10748

If [vkCmdSetPrimitiveTopologyEXT](#) set [primitiveTopology](#) to [VK_PRIMITIVE_TOPOLOGY_POINT_LIST](#) prior to this drawing command, both a [VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT](#) and [VK_SHADER_STAGE_GEOMETRY_BIT](#) stage are not bound, then the [Vertex Execution Model](#) **must** have a [PointSize](#) decorated variable that

is statically written to

- VUID-vkCmdDrawIndexed-pStrides-04913

If the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE` dynamic state enabled, but without the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, then `vkCmdBindVertexBuffers2EXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this draw command, and the `pStrides` parameter of `vkCmdBindVertexBuffers2EXT` **must** not be `NULL`

- VUID-vkCmdDrawIndexed-None-04914

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then `vkCmdSetVertexInputEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this draw command

- VUID-vkCmdDrawIndexed-Input-07939

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then all variables with the `Input` storage class decorated with `Location` in the `Vertex Execution Model OpEntryPoint` **must** contain a location in `VkVertexInputAttributeDescription2EXT::location`

- VUID-vkCmdDrawIndexed-Input-08734

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then the numeric type associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be the same as `VkVertexInputAttributeDescription2EXT::format`

- VUID-vkCmdDrawIndexed-format-08936

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and `VkVertexInputAttributeDescription2EXT::format` has a 64-bit component, then the scalar width associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be 64-bit

- VUID-vkCmdDrawIndexed-format-08937

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and the scalar width associated with a `Location` decorated `Input` variable in the `Vertex Execution Model OpEntryPoint` is 64-bit, then the corresponding `VkVertexInputAttributeDescription2EXT::format` **must** have a 64-bit component

- VUID-vkCmdDrawIndexed-None-09203

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and `VkVertexInputAttributeDescription2EXT::format` has a 64-bit component, then all `Input` variables at the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** not use components that are not present in the format

- VUID-vkCmdDrawIndexed-None-04875

If the bound graphics pipeline state was created with both a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage and the `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT` dynamic state enabled, and the `current value`

of `primitiveTopology` is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, then `vkCmdSetPatchControlPointsEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-04879

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled then `vkCmdSetPrimitiveRestartEnableEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexed-None-09637

If the `input assembly` is `VK_PRIMITIVE_TOPOLOGY_POINT_LIST`, `VK_PRIMITIVE_TOPOLOGY_LINE_LIST`, `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST`, `VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY`, or `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY`, the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled, then `vkCmdSetPrimitiveRestartEnableEXT` **must** be `VK_FALSE`

- VUID-vkCmdDrawIndexed-None-10909

If the `input assembly` is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled then `vkCmdSetPrimitiveRestartEnableEXT` **must** be `VK_FALSE`

- VUID-vkCmdDrawIndexed-None-07312

A valid index buffer **must** be bound

- VUID-vkCmdDrawIndexed-pNext-09461

If the bound graphics pipeline state was created with `VkPipelineVertexInputDivisorStateCreateInfo` in the `pNext` chain of `VkGraphicsPipelineCreateInfo::pVertexInputState`, any member of `VkPipelineVertexInputDivisorStateCreateInfo::pVertexBindingDivisors` has a value other than `1` in `divisor`, and `VkPhysicalDeviceVertexAttributeDivisorProperties::supportsNonZeroFirstInstance` is `VK_FALSE`, then `firstInstance` **must** be `0`

- VUID-vkCmdDrawIndexed-None-09462

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, any member of the `pVertexBindingDescriptions` parameter to the `vkCmdSetVertexInputEXT` call that sets this dynamic state has a value other than `1` in `divisor`, and `VkPhysicalDeviceVertexAttributeDivisorProperties::supportsNonZeroFirstInstance` is `VK_FALSE`, then `firstInstance` **must** be `0`

- VUID-vkCmdDrawIndexed-robustBufferAccess2-08798

If the `robustBufferAccess2` feature is not enabled, $(indexSize \times (firstIndex + indexCount) + offset)$ **must** be less than or equal to the size of the bound index buffer, with `indexSize` being based on the type specified by `indexType`, where the index buffer, `indexType`, and `offset` are specified via `vkCmdBindIndexBuffer`

Valid Usage (Implicit)

- VUID-vkCmdDrawIndexed-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdDrawIndexed-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdDrawIndexed-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdDrawIndexed-renderpass
This command **must** only be called inside of a render pass instance

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Inside	VK_QUEUE_GRAPHICS_BIT	Action

To record a non-indexed indirect drawing command, call:

```
// Provided by VK_VERSION_1_0
void vkCmdDrawIndirect(
    VkCommandBuffer          commandBuffer,
    VkBuffer                 buffer,
    VkDeviceSize             offset,
    uint32_t                 drawCount,
    uint32_t                 stride);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `buffer` is the buffer containing draw parameters.
- `offset` is the byte offset into `buffer` where parameters begin.
- `drawCount` is the number of draws to execute, and **can** be zero.
- `stride` is the byte stride between successive sets of draw parameters.

`vkCmdDrawIndirect` behaves similarly to `vkCmdDraw` except that the parameters are read by the device from a buffer during execution. `drawCount` draws are executed by the command, with parameters taken from `buffer` starting at `offset` and increasing by `stride` bytes for each successive draw. The parameters of each draw are encoded in an array of `VkDrawIndirectCommand` structures. If `drawCount` is less than or equal to one, `stride` is ignored.

Valid Usage

- VUID-vkCmdDrawIndirect-magFilter-04553

If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDrawIndirect-magFilter-09598

If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`

- VUID-vkCmdDrawIndirect-mipmapMode-04770

If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDrawIndirect-mipmapMode-09599

If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`

- VUID-vkCmdDrawIndirect-unnormalizedCoordinates-09635

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `levelCount` and `layerCount` **must** be 1

- VUID-vkCmdDrawIndirect-None-08609

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `viewType` **must** be `VK_IMAGE_VIEW_TYPE_1D` or `VK_IMAGE_VIEW_TYPE_2D`

- VUID-vkCmdDrawIndirect-None-08610

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name

- VUID-vkCmdDrawIndirect-None-08611
If a [VkSampler](#) created with [unnormalizedCoordinates](#) equal to [VK_TRUE](#) is used to sample a [VkImageView](#) as a result of this command, then the sampler **must** not be used with any of the SPIR-V [OpImageSample*](#) or [OpImageSparseSample*](#) instructions that includes a LOD bias or any offset values
- VUID-vkCmdDrawIndirect-aspectMask-06478
If a [VkImageView](#) is sampled with [depth comparison](#), the image view **must** have been created with an [aspectMask](#) that contains [VK_IMAGE_ASPECT_DEPTH_BIT](#)
- VUID-vkCmdDrawIndirect-None-02691
If a [VkImageView](#) is accessed using atomic operations as a result of this command, then the image view's [format](#) [features](#) **must** contain [VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT](#)
- VUID-vkCmdDrawIndirect-None-07888
If a [VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER](#) descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's [format](#) [features](#) **must** contain [VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT](#)
- VUID-vkCmdDrawIndirect-None-02692
If a [VkImageView](#) is sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command, then the image view's [format](#) [features](#) **must** contain [VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT](#)
- VUID-vkCmdDrawIndirect-None-02693
If the [VK_EXT_filter_cubic](#) extension is not enabled and any [VkImageView](#) is sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command, it **must** not have a [VkImageViewType](#) of [VK_IMAGE_VIEW_TYPE_3D](#), [VK_IMAGE_VIEW_TYPE_CUBE](#), or [VK_IMAGE_VIEW_TYPE_CUBE_ARRAY](#)
- VUID-vkCmdDrawIndirect-filterCubic-02694
Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)
- VUID-vkCmdDrawIndirect-filterCubicMinmax-02695
Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) with a reduction mode of either [VK_SAMPLER_REDUCTION_MODE_MIN](#) or [VK_SAMPLER_REDUCTION_MODE_MAX](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering together with minmax filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)
- VUID-vkCmdDrawIndirect-None-08600
For each set *n* that is statically used by a [bound shader](#), a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for set *n*, with the [VkPipelineLayout](#) used to create the current [VkPipeline](#) , as described in [Pipeline Layout Compatibility](#)
- VUID-vkCmdDrawIndirect-None-08601
For each push constant that is statically used by a [bound shader](#), a push constant value **must** have been set for the same pipeline bind point, with a [VkPipelineLayout](#) that is

compatible for push constants, with the [VkPipelineLayout](#) used to create the current [VkPipeline](#) , as described in [Pipeline Layout Compatibility](#)

- VUID-vkCmdDrawIndirect-None-10068

For each array of resources that is used by a [bound shader](#), the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command

- VUID-vkCmdDrawIndirect-None-08114

Descriptors in each bound descriptor set, specified via [vkCmdBindDescriptorSets](#), **must** be valid as described by [descriptor validity](#) if they are statically used by a [bound shader](#)

- VUID-vkCmdDrawIndirect-imageLayout-00344

If an image descriptor is accessed by a shader, the [VkImageLayout](#) **must** match the subresource accessible from the [VkImageView](#) as defined by the [image layout matching rules](#)

- VUID-vkCmdDrawIndirect-None-08606

A valid pipeline **must** be bound to the pipeline bind point used by this command

- VUID-vkCmdDrawIndirect-None-08608

There **must** not have been any calls to dynamic state setting commands for any state specified statically in the [VkPipeline](#) object bound to the pipeline bind point used by this command, since that pipeline was bound

- VUID-vkCmdDrawIndirect-uniformBuffers-06935

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a uniform buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndirect-storageBuffers-06936

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a storage buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndirect-commandBuffer-02707

If [commandBuffer](#) is an unprotected command buffer and [protectedNoFault](#) is not supported, any resource accessed by [bound shaders](#) **must** not be a protected resource

- VUID-vkCmdDrawIndirect-viewType-07752

If a [VkImageView](#) is accessed as a result of this command, then the image view's [viewType](#) **must** match the [Dim](#) operand of the [OpTypeImage](#) as described in [Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types](#)

- VUID-vkCmdDrawIndirect-format-07753

If a [VkImageView](#) or [VkBufferView](#) is accessed as a result of this command, then the [numeric type](#) of the view's [format](#) and the [Sampled Type](#) operand of the [OpTypeImage](#) **must** match

- VUID-vkCmdDrawIndirect-OpImageWrite-08795

If a [VkImageView](#) is accessed using [OpImageWrite](#) as a result of this command, then the [Type](#) of the [Texel](#) operand of that instruction **must** have at least as many components as

the image view's format

- VUID-vkCmdDrawIndirect-OpImageWrite-04469

If a `VkBufferView` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDrawIndirect-SampledType-04470

If a `VkImageView` with a `VkFormat` that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64

- VUID-vkCmdDrawIndirect-SampledType-04471

If a `VkImageView` with a `VkFormat` that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32

- VUID-vkCmdDrawIndirect-SampledType-04472

If a `VkBufferView` with a `VkFormat` that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64

- VUID-vkCmdDrawIndirect-SampledType-04473

If a `VkBufferView` with a `VkFormat` that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32

- VUID-vkCmdDrawIndirect-sparseImageInt64Atomics-04474

If the `sparseImageInt64Atomics` feature is not enabled, `VkImage` objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command

- VUID-vkCmdDrawIndirect-sparseImageInt64Atomics-04475

If the `sparseImageInt64Atomics` feature is not enabled, `VkBuffer` objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command

- VUID-vkCmdDrawIndirect-None-07288

Any shader invocation executed by this command **must** `terminate`

- VUID-vkCmdDrawIndirect-None-09600

If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, all image subresources identified by that descriptor **must** be in the image layout identified when the descriptor was written

- VUID-vkCmdDrawIndirect-renderPass-02684

The current render pass **must** be `compatible` with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`

- VUID-vkCmdDrawIndirect-subpass-02685

The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to

VK_PIPELINE_BIND_POINT_GRAPHICS

- VUID-vkCmdDrawIndirect-None-07748

If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set

- VUID-vkCmdDrawIndirect-OpTypeImage-07468

If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass

- VUID-vkCmdDrawIndirect-None-07469

Input attachment views accessed in a subpass **must** be created with the same `VkFormat` as the corresponding subpass definition, and be created with a `VkImageView` that is compatible with the attachment referenced by the subpass' `pInputAttachments[InputAttachmentIndex]` in the bound `VkFramebuffer` as specified by [Fragment Input Attachment Compatibility](#)

- VUID-vkCmdDrawIndirect-None-06537

Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirect-None-10795

If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirect-None-10796

If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirect-None-10797

If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirect-None-06539

If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment

- VUID-vkCmdDrawIndirect-None-06886

If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, `depth writes` **must** be disabled

- VUID-vkCmdDrawIndirect-None-06887

If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back `writeMask` are not zero, and stencil test is enabled, `all stencil ops` **must** be `VK_STENCIL_OP_KEEP`

- VUID-vkCmdDrawIndirect-None-07831
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled then `vkCmdSetViewport` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07832
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled then `vkCmdSetScissor` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-08617
If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `effective rasterization input topology` is in line topology class, then `vkCmdSetLineWidth` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07834
If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthBiasEnable` is `VK_TRUE`, then `vkCmdSetDepthBias` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07835
If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_BLEND_CONSTANTS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and an active color attachment `current value` of `blendEnable` is `VK_TRUE` with a blend equations where any `VkBlendFactor` member is `VK_BLEND_FACTOR_CONSTANT_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR`, `VK_BLEND_FACTOR_CONSTANT_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA`, then `vkCmdSetBlendConstants` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07836
If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BOUNDS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthBoundsTestEnable` is `VK_TRUE`, then `vkCmdSetDepthBounds` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07837
If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilCompareMask` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-07838
If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `stencilTestEnable` is `VK_TRUE`,

then `vkCmdSetStencilWriteMask` **must** have been called and not subsequently *invalidated* in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07839

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_REFERENCE` dynamic state enabled, the *current value* of `rasterizerDiscardEnable` is `VK_FALSE`, the *current value* of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilReference` **must** have been called and not subsequently *invalidated* in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-maxMultiviewInstanceIndex-02688

If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to `VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex`

- VUID-vkCmdDrawIndirect-sampleLocationsEnable-02689

If the bound graphics pipeline was created with `VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable` set to `VK_TRUE`, then the active depth attachment **must** have been created with the `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` bit set

- VUID-vkCmdDrawIndirect-None-06666

If the `VK_EXT_sample_locations` extension is enabled, a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` dynamic state enabled, the *current value* of `rasterizerDiscardEnable` is `VK_FALSE`, and the *current value* of `sampleLocationsEnable` is `VK_TRUE`, then `vkCmdSetSampleLocationsEXT` **must** have been called and not subsequently *invalidated* in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07840

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_CULL_MODE` dynamic state enabled, and the *current value* of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetCullModeEXT` **must** have been called and not subsequently *invalidated* in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07841

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_FRONT_FACE` dynamic state enabled, and the *current value* of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetFrontFaceEXT` **must** have been called and not subsequently *invalidated* in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07843

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE` dynamic state enabled, and the *current value* of `rasterizerDiscardEnable` is `VK_FALSE`, `vkCmdSetDepthTestEnableEXT` **must** have been called and not subsequently *invalidated* in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07844

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` dynamic state enabled, and the *current value* of `rasterizerDiscardEnable` is `VK_FALSE`, and the *current value* of `depthTestEnable` is `VK_TRUE`, then `vkCmdSetDepthWriteEnableEXT` **must** have been called and not subsequently

invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07845

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP` dynamic state enabled, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, and the **current value** of `depthTestEnable` is `VK_TRUE`, then `vkCmdSetDepthCompareOpEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07846

If the `depthBounds` feature is enabled, a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE` dynamic state enabled, and the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetDepthBoundsTestEnableEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07847

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE` dynamic state enabled, and the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetStencilTestEnableEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-07848

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_OP` dynamic state enabled, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, the **current value** of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilOpEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-viewportCount-03417

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, then `vkCmdSetViewportWithCountEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-scissorCount-03418

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` dynamic state enabled, then `vkCmdSetScissorWithCountEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing

- VUID-vkCmdDrawIndirect-viewportCount-03419

If a graphics pipeline is bound which was created with both the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic states enabled, then the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** match the `scissorCount` parameter of `vkCmdSetScissorWithCountEXT`

- VUID-vkCmdDrawIndirect-None-04876

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` dynamic state enabled, then `vkCmdSetRasterizerDiscardEnableEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-04877

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetDepthBiasEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-logicOp-04878

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LOGIC_OP_EXT` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `logicOpEnable` is `VK_TRUE`, then `vkCmdSetLogicOpEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-primitiveFragmentShadingRateWithMultipleViewports-04552

If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the `PrimitiveShadingRateKHR` built-in, then `vkCmdSetViewportWithCountEXT` **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be 1

- VUID-vkCmdDrawIndirect-blendEnable-04727

If a graphics pipeline is bound which was created with `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then for each color attachment, if the corresponding image view's `format` `features` do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the corresponding `current value` of `blendEnable` **must** be `VK_FALSE`

- VUID-vkCmdDrawIndirect-None-08644

If a graphics pipeline is bound, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then the `current value` of `rasterizationSamples` **must** be the same as the current color and/or depth/stencil attachments

- VUID-vkCmdDrawIndirect-maxFragmentDualSrcAttachments-09239

If `blending` is enabled for any attachment where either the source or destination blend factors for that attachment `use the secondary color input`, the maximum value of `Location` for any output attachment `statically used` in the `Fragment Execution Model` executed by this command **must** be less than `maxFragmentDualSrcAttachments`

- VUID-vkCmdDrawIndirect-None-04007

All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or `VK_NULL_HANDLE` buffers bound

- VUID-vkCmdDrawIndirect-None-04008

If the `nullDescriptor` feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be `VK_NULL_HANDLE`

- VUID-vkCmdDrawIndirect-None-02721
If the `robustBufferAccess` feature is not enabled, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in [Vertex Input Description](#)
- VUID-vkCmdDrawIndirect-format-10389
For each vertex attribute accessed by this command, if its `VkVertexInputAttributeDescription::format` or `VkVertexInputAttributeDescription2EXT::format` is a `packed format`, the value of `attribAddress`, calculated as described in [Vertex Input Calculation](#), **must** be a multiple of the `size of the format`
- VUID-vkCmdDrawIndirect-format-10390
For each vertex attribute accessed by this command, if its `VkVertexInputAttributeDescription::format` or `VkVertexInputAttributeDescription2EXT::format` is not a `packed format`, the value of `attribAddress`, calculated as described in [Vertex Input Calculation](#), **must** be a multiple of the `component size of the format`
- VUID-vkCmdDrawIndirect-None-07842
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then `vkCmdSetPrimitiveTopologyEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-dynamicPrimitiveTopologyUnrestricted-07500
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then the `primitiveTopology` parameter of `vkCmdSetPrimitiveTopologyEXT` **must** be of the same `topology class` as the pipeline `VkPipelineInputAssemblyStateCreateInfo::topology` state
- VUID-vkCmdDrawIndirect-primitiveTopology-10286
If a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage is bound, then the `current value` of `primitiveTopology` **must** be `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST` prior to this drawing command
- VUID-vkCmdDrawIndirect-primitiveTopology-10747
If `vkCmdSetPrimitiveTopologyEXT` set `primitiveTopology` to `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST` prior to this drawing command, then a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage **must** be bound
- VUID-vkCmdDrawIndirect-primitiveTopology-10748
If `vkCmdSetPrimitiveTopologyEXT` set `primitiveTopology` to `VK_PRIMITIVE_TOPOLOGY_POINT_LIST` prior to this drawing command, both a `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` and `VK_SHADER_STAGE_GEOMETRY_BIT` stage are not bound, then the `Vertex Execution Model` **must** have a `PointSize` decorated variable that is statically written to
- VUID-vkCmdDrawIndirect-pStrides-04913
If the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE` dynamic state enabled, but without the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, then `vkCmdBindVertexBuffers2EXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this draw command, and the `pStrides` parameter of `vkCmdBindVertexBuffers2EXT` **must** not be `NULL`

- VUID-vkCmdDrawIndirect-None-04914
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then `vkCmdSetVertexInputEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this draw command
- VUID-vkCmdDrawIndirect-Input-07939
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then all variables with the `Input` storage class decorated with `Location` in the `Vertex Execution Model OpEntryPoint` **must** contain a location in `VkVertexInputAttributeDescription2EXT::location`
- VUID-vkCmdDrawIndirect-Input-08734
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then the numeric type associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be the same as `VkVertexInputAttributeDescription2EXT::format`
- VUID-vkCmdDrawIndirect-format-08936
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and `VkVertexInputAttributeDescription2EXT::format` has a 64-bit component, then the scalar width associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be 64-bit
- VUID-vkCmdDrawIndirect-format-08937
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and the scalar width associated with a `Location` decorated `Input` variable in the `Vertex Execution Model OpEntryPoint` is 64-bit, then the corresponding `VkVertexInputAttributeDescription2EXT::format` **must** have a 64-bit component
- VUID-vkCmdDrawIndirect-None-09203
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and `VkVertexInputAttributeDescription2EXT::format` has a 64-bit component, then all `Input` variables at the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** not use components that are not present in the format
- VUID-vkCmdDrawIndirect-None-04875
If the bound graphics pipeline state was created with both a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage and the `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT` dynamic state enabled, and the `current value` of `primitiveTopology` is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, then `vkCmdSetPatchControlPointsEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirect-None-04879
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled then `vkCmdSetPrimitiveRestartEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirect-None-09637
If the **input assembly** is **VK_PRIMITIVE_TOPOLOGY_POINT_LIST**, **VK_PRIMITIVE_TOPOLOGY_LINE_LIST**, **VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST**, **VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY**, or **VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY**, the bound graphics pipeline state was created with the **VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE** dynamic state enabled, then **vkCmdSetPrimitiveRestartEnableEXT** **must** be **VK_FALSE**
- VUID-vkCmdDrawIndirect-None-10909
If the **input assembly** is **VK_PRIMITIVE_TOPOLOGY_PATCH_LIST**, the bound graphics pipeline state was created with the **VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE** dynamic state enabled then **vkCmdSetPrimitiveRestartEnableEXT** **must** be **VK_FALSE**
- VUID-vkCmdDrawIndirect-buffer-02708
If **buffer** is non-sparse then it **must** be bound completely and contiguously to a single **VkDeviceMemory** object
- VUID-vkCmdDrawIndirect-buffer-02709
buffer **must** have been created with the **VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT** bit set
- VUID-vkCmdDrawIndirect-offset-02710
offset **must** be a multiple of 4
- VUID-vkCmdDrawIndirect-commandBuffer-02711
commandBuffer **must** not be a protected command buffer
- VUID-vkCmdDrawIndirect-drawCount-02718
If the **multiDrawIndirect** feature is not enabled, **drawCount** **must** be 0 or 1
- VUID-vkCmdDrawIndirect-drawCount-02719
drawCount **must** be less than or equal to **VkPhysicalDeviceLimits::maxDrawIndirectCount**
- VUID-vkCmdDrawIndirect-drawCount-00476
If **drawCount** is greater than 1, **stride** **must** be a multiple of 4 and **must** be greater than or equal to **sizeof(VkDrawIndirectCommand)**
- VUID-vkCmdDrawIndirect-drawCount-00487
If **drawCount** is equal to 1, (**offset** + **sizeof(VkDrawIndirectCommand)**) **must** be less than or equal to the size of **buffer**
- VUID-vkCmdDrawIndirect-drawCount-00488
If **drawCount** is greater than 1, (**stride** × (**drawCount** - 1) + **offset** + **sizeof(VkDrawIndirectCommand)**) **must** be less than or equal to the size of **buffer**

Valid Usage (Implicit)

- VUID-vkCmdDrawIndirect-commandBuffer-parameter
commandBuffer **must** be a valid **VkCommandBuffer** handle
- VUID-vkCmdDrawIndirect-buffer-parameter
buffer **must** be a valid **VkBuffer** handle
- VUID-vkCmdDrawIndirect-commandBuffer-recording

`commandBuffer` **must** be in the `recording` state

- VUID-vkCmdDrawIndirect-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

- VUID-vkCmdDrawIndirect-renderpass

This command **must** only be called inside of a render pass instance

- VUID-vkCmdDrawIndirect-commonparent

Both of `buffer`, and `commandBuffer` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Inside	VK_QUEUE_GRAPHICS_BIT	Action

The `VkDrawIndirectCommand` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDrawIndirectCommand {
    uint32_t    vertexCount;
    uint32_t    instanceCount;
    uint32_t    firstVertex;
    uint32_t    firstInstance;
} VkDrawIndirectCommand;
```

- `vertexCount` is the number of vertices to draw.
- `instanceCount` is the number of instances to draw.
- `firstVertex` is the index of the first vertex to draw.
- `firstInstance` is the instance ID of the first instance to draw.

The members of `VkDrawIndirectCommand` have the same meaning as the similarly named parameters of `vkCmdDraw`.

Valid Usage

- VUID-VkDrawIndirectCommand-pNext-09461

If the bound graphics pipeline state was created with `VkPipelineVertexInputDivisorStateCreateInfo` in the `pNext` chain of `VkGraphicsPipelineCreateInfo::pVertexInputState`, any member of `VkPipelineVertexInputDivisorStateCreateInfo::pVertexBindingDivisors` has a value other than `1` in `divisor`, and `VkPhysicalDeviceVertexAttributeDivisorProperties::supportsNonZeroFirstInstance` is `VK_FALSE`, then `firstInstance` **must** be `0`

- VUID-VkDrawIndirectCommand-None-09462

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, any member of the `pVertexBindingDescriptions` parameter to the `vkCmdSetVertexInputEXT` call that sets this dynamic state has a value other than `1` in `divisor`, and `VkPhysicalDeviceVertexAttributeDivisorProperties::supportsNonZeroFirstInstance` is `VK_FALSE`, then `firstInstance` **must** be `0`

- VUID-VkDrawIndirectCommand-firstInstance-00501

If the `drawIndirectFirstInstance` feature is not enabled, `firstInstance` **must** be `0`

To record a non-indexed draw call with a draw call count sourced from a buffer, call:

```
// Provided by VK_VERSION_1_2
void vkCmdDrawIndirectCount(
    VkCommandBuffer          commandBuffer,
    VkBuffer                  buffer,
    VkDeviceSize              offset,
    VkBuffer                  countBuffer,
    VkDeviceSize              countBufferOffset,
    uint32_t                  maxDrawCount,
    uint32_t                  stride);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `buffer` is the buffer containing draw parameters.
- `offset` is the byte offset into `buffer` where parameters begin.
- `countBuffer` is the buffer containing the draw count.
- `countBufferOffset` is the byte offset into `countBuffer` where the draw count begins.
- `maxDrawCount` specifies the maximum number of draws that will be executed. The actual number of executed draw calls is the minimum of the count specified in `countBuffer` and `maxDrawCount`.
- `stride` is the byte stride between successive sets of draw parameters.

`vkCmdDrawIndirectCount` behaves similarly to `vkCmdDrawIndirect` except that the draw count is read by the device from a buffer during execution. The command will read an unsigned 32-bit integer from `countBuffer` located at `countBufferOffset` and use this as the draw count.

Valid Usage

- VUID-vkCmdDrawIndirectCount-magFilter-04553

If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDrawIndirectCount-magFilter-09598

If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`

- VUID-vkCmdDrawIndirectCount-mipmapMode-04770

If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDrawIndirectCount-mipmapMode-09599

If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`

- VUID-vkCmdDrawIndirectCount-unnormalizedCoordinates-09635

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `levelCount` and `layerCount` **must** be 1

- VUID-vkCmdDrawIndirectCount-None-08609

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `viewType` **must** be `VK_IMAGE_VIEW_TYPE_1D` or `VK_IMAGE_VIEW_TYPE_2D`

- VUID-vkCmdDrawIndirectCount-None-08610

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name

- VUID-vkCmdDrawIndirectCount-None-08611

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values

- VUID-vkCmdDrawIndirectCount-aspectMask-06478
If a [VkImageView](#) is sampled with [depth comparison](#), the image view **must** have been created with an [aspectMask](#) that contains [VK_IMAGE_ASPECT_DEPTH_BIT](#)
- VUID-vkCmdDrawIndirectCount-None-02691
If a [VkImageView](#) is accessed using atomic operations as a result of this command, then the image view's [format](#) [features](#) **must** contain [VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT](#)
- VUID-vkCmdDrawIndirectCount-None-07888
If a [VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER](#) descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's [format](#) [features](#) **must** contain [VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT](#)
- VUID-vkCmdDrawIndirectCount-None-02692
If a [VkImageView](#) is sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command, then the image view's [format](#) [features](#) **must** contain [VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT](#)
- VUID-vkCmdDrawIndirectCount-None-02693
If the [VK_EXT_filter_cubic](#) extension is not enabled and any [VkImageView](#) is sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command, it **must** not have a [VkImageViewType](#) of [VK_IMAGE_VIEW_TYPE_3D](#), [VK_IMAGE_VIEW_TYPE_CUBE](#), or [VK_IMAGE_VIEW_TYPE_CUBE_ARRAY](#)
- VUID-vkCmdDrawIndirectCount-filterCubic-02694
Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)
- VUID-vkCmdDrawIndirectCount-filterCubicMinmax-02695
Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) with a reduction mode of either [VK_SAMPLER_REDUCTION_MODE_MIN](#) or [VK_SAMPLER_REDUCTION_MODE_MAX](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering together with minmax filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)
- VUID-vkCmdDrawIndirectCount-None-08600
For each set *n* that is statically used by a [bound shader](#), a descriptor set **must** have been bound to *n* at the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for set *n*, with the [VkPipelineLayout](#) used to create the current [VkPipeline](#) , as described in [Pipeline Layout Compatibility](#)
- VUID-vkCmdDrawIndirectCount-None-08601
For each push constant that is statically used by a [bound shader](#), a push constant value **must** have been set for the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for push constants, with the [VkPipelineLayout](#) used to create the current [VkPipeline](#) , as described in [Pipeline Layout Compatibility](#)
- VUID-vkCmdDrawIndirectCount-None-10068
For each array of resources that is used by a [bound shader](#), the indices used to access members of the array **must** be less than the descriptor count for the identified binding in

the descriptor sets used by this command

- VUID-vkCmdDrawIndirectCount-None-08114

Descriptors in each bound descriptor set, specified via [vkCmdBindDescriptorSets](#), **must** be valid as described by [descriptor validity](#) if they are statically used by a [bound shader](#)

- VUID-vkCmdDrawIndirectCount-imageLayout-00344

If an image descriptor is accessed by a shader, the [VkImageLayout](#) **must** match the subresource accessible from the [VkImageView](#) as defined by the [image layout matching rules](#)

- VUID-vkCmdDrawIndirectCount-None-08606

A valid pipeline **must** be bound to the pipeline bind point used by this command

- VUID-vkCmdDrawIndirectCount-None-08608

There **must** not have been any calls to dynamic state setting commands for any state specified statically in the [VkPipeline](#) object bound to the pipeline bind point used by this command, since that pipeline was bound

- VUID-vkCmdDrawIndirectCount-uniformBuffers-06935

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a uniform buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndirectCount-storageBuffers-06936

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a storage buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndirectCount-commandBuffer-02707

If [commandBuffer](#) is an unprotected command buffer and [protectedNoFault](#) is not supported, any resource accessed by [bound shaders](#) **must** not be a protected resource

- VUID-vkCmdDrawIndirectCount-viewType-07752

If a [VkImageView](#) is accessed as a result of this command, then the image view's [viewType](#) **must** match the [Dim](#) operand of the [OpTypeImage](#) as described in [Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types](#)

- VUID-vkCmdDrawIndirectCount-format-07753

If a [VkImageView](#) or [VkBufferView](#) is accessed as a result of this command, then the [numeric type](#) of the view's [format](#) and the [Sampled Type](#) operand of the [OpTypeImage](#) **must** match

- VUID-vkCmdDrawIndirectCount-OpImageWrite-08795

If a [VkImageView](#) is accessed using [OpImageWrite](#) as a result of this command, then the [Type](#) of the [Texel](#) operand of that instruction **must** have at least as many components as the image view's format

- VUID-vkCmdDrawIndirectCount-OpImageWrite-04469

If a [VkBufferView](#) is accessed using [OpImageWrite](#) as a result of this command, then the [Type](#) of the [Texel](#) operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDrawIndirectCount-SampledType-04470
If a [VkImageView](#) with a [VkFormat](#) that has a 64-bit component width is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 64
- VUID-vkCmdDrawIndirectCount-SampledType-04471
If a [VkImageView](#) with a [VkFormat](#) that has a component width less than 64-bit is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 32
- VUID-vkCmdDrawIndirectCount-SampledType-04472
If a [VkBufferView](#) with a [VkFormat](#) that has a 64-bit component width is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 64
- VUID-vkCmdDrawIndirectCount-SampledType-04473
If a [VkBufferView](#) with a [VkFormat](#) that has a component width less than 64-bit is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 32
- VUID-vkCmdDrawIndirectCount-sparseImageInt64Atomics-04474
If the [sparseImageInt64Atomics](#) feature is not enabled, [VkImage](#) objects created with the [VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT](#) flag **must** not be accessed by atomic instructions through an [OpTypeImage](#) with a [SampledType](#) with a [Width](#) of 64 by this command
- VUID-vkCmdDrawIndirectCount-sparseImageInt64Atomics-04475
If the [sparseImageInt64Atomics](#) feature is not enabled, [VkBuffer](#) objects created with the [VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT](#) flag **must** not be accessed by atomic instructions through an [OpTypeImage](#) with a [SampledType](#) with a [Width](#) of 64 by this command
- VUID-vkCmdDrawIndirectCount-None-07288
Any shader invocation executed by this command **must** [terminate](#)
- VUID-vkCmdDrawIndirectCount-None-09600
If a descriptor with type equal to any of [VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE](#), [VK_DESCRIPTOR_TYPE_STORAGE_IMAGE](#), or [VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT](#) is accessed as a result of this command, all image subresources identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawIndirectCount-renderPass-02684
The current render pass **must** be [compatible](#) with the [renderPass](#) member of the [VkGraphicsPipelineCreateInfo](#) structure specified when creating the [VkPipeline](#) bound to [VK_PIPELINE_BIND_POINT_GRAPHICS](#)
- VUID-vkCmdDrawIndirectCount-subpass-02685
The subpass index of the current render pass **must** be equal to the [subpass](#) member of the [VkGraphicsPipelineCreateInfo](#) structure specified when creating the [VkPipeline](#) bound to [VK_PIPELINE_BIND_POINT_GRAPHICS](#)
- VUID-vkCmdDrawIndirectCount-None-07748
If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawIndirectCount-OpTypeImage-07468

If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass

- VUID-vkCmdDrawIndirectCount-None-07469

Input attachment views accessed in a subpass **must** be created with the same `VkFormat` as the corresponding subpass definition, and be created with a `VkImageView` that is compatible with the attachment referenced by the subpass' `pInputAttachments` [`InputAttachmentIndex`] in the bound `VkFramebuffer` as specified by [Fragment Input Attachment Compatibility](#)

- VUID-vkCmdDrawIndirectCount-None-06537

Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirectCount-None-10795

If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirectCount-None-10796

If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirectCount-None-10797

If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndirectCount-None-06539

If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment

- VUID-vkCmdDrawIndirectCount-None-06886

If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, [depth writes](#) **must** be disabled

- VUID-vkCmdDrawIndirectCount-None-06887

If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back `writeMask` are not zero, and stencil test is enabled, [all stencil ops](#) **must** be `VK_STENCIL_OP_KEEP`

- VUID-vkCmdDrawIndirectCount-None-07831

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled then `vkCmdSetViewport` **must** have been called and not subsequently [invalidated](#) in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07832

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled then `vkCmdSetScissor` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-08617

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `effective rasterization input topology` is in line topology class, then `vkCmdSetLineWidth` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07834

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthBiasEnable` is `VK_TRUE`, then `vkCmdSetDepthBias` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07835

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_BLEND_CONSTANTS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and an active color attachment `current value` of `blendEnable` is `VK_TRUE` with a blend equations where any `VkBlendFactor` member is `VK_BLEND_FACTOR_CONSTANT_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR`, `VK_BLEND_FACTOR_CONSTANT_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA`, then `vkCmdSetBlendConstants` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07836

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BOUNDS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthBoundsTestEnable` is `VK_TRUE`, then `vkCmdSetDepthBounds` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07837

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilCompareMask` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07838

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilWriteMask` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07839

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_REFERENCE` dynamic state enabled, the `current value` of and

`rasterizerDiscardEnable` is `VK_FALSE`, the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilReference` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-maxMultiviewInstanceIndex-02688

If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to `VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex`

- VUID-vkCmdDrawIndirectCount-sampleLocationsEnable-02689

If the bound graphics pipeline was created with `VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable` set to `VK_TRUE`, then the active depth attachment **must** have been created with the `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` bit set

- VUID-vkCmdDrawIndirectCount-None-06666

If the `VK_EXT_sample_locations` extension is enabled, a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `sampleLocationsEnable` is `VK_TRUE`, then `vkCmdSetSampleLocationsEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07840

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_CULL_MODE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetCullModeEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07841

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_FRONT_FACE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetFrontFaceEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07843

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, `vkCmdSetDepthTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07844

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthTestEnable` is `VK_TRUE`, then `vkCmdSetDepthWriteEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07845

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthTestEnable` is `VK_TRUE`,

then `vkCmdSetDepthCompareOpEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07846

If the `depthBounds` feature is enabled, a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetDepthBoundsTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07847

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetStencilTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-07848

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_OP` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilOpEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-viewportCount-03417

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, then `vkCmdSetViewportWithCountEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-scissorCount-03418

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` dynamic state enabled, then `vkCmdSetScissorWithCountEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing

- VUID-vkCmdDrawIndirectCount-viewportCount-03419

If a graphics pipeline is bound which was created with both the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic states enabled, then the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** match the `scissorCount` parameter of `vkCmdSetScissorWithCountEXT`

- VUID-vkCmdDrawIndirectCount-None-04876

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` dynamic state enabled, then `vkCmdSetRasterizerDiscardEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-04877

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetDepthBiasEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this

drawing command

- VUID-vkCmdDrawIndirectCount-logicOp-04878

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LOGIC_OP_EXT` dynamic state enabled, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, and the **current value** of `logicOpEnable` is `VK_TRUE`, then `vkCmdSetLogicOpEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-primitiveFragmentShadingRateWithMultipleViewports-04552

If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the `PrimitiveShadingRateKHR` built-in, then `vkCmdSetViewportWithCountEXT` **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be 1

- VUID-vkCmdDrawIndirectCount-blendEnable-04727

If a graphics pipeline is bound which was created with `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic state enabled, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, then for each color attachment, if the corresponding image view's **format features** do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the corresponding **current value** of `blendEnable` **must** be `VK_FALSE`

- VUID-vkCmdDrawIndirectCount-None-08644

If a graphics pipeline is bound, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`,

then the **current value** of `rasterizationSamples` **must** be the same as the current color and/or depth/stencil attachments

- VUID-vkCmdDrawIndirectCount-maxFragmentDualSrcAttachments-09239

If **blending** is enabled for any attachment where either the source or destination blend factors for that attachment **use the secondary color input**, the maximum value of `Location` for any output attachment **statically used** in the `Fragment Execution Model` executed by this command **must** be less than `maxFragmentDualSrcAttachments`

- VUID-vkCmdDrawIndirectCount-None-04007

All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or `VK_NULL_HANDLE` buffers bound

- VUID-vkCmdDrawIndirectCount-None-04008

If the `nullDescriptor` feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be `VK_NULL_HANDLE`

- VUID-vkCmdDrawIndirectCount-None-02721

If the `robustBufferAccess` feature is not enabled, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in [Vertex Input Description](#)

- VUID-vkCmdDrawIndirectCount-format-10389
For each vertex attribute accessed by this command, if its `VkVertexInputAttributeDescription::format` or `VkVertexInputAttributeDescription2EXT::format` is a **packed format**, the value of `attribAddress`, calculated as described in **Vertex Input Calculation**, **must** be a multiple of the **size of the format**
- VUID-vkCmdDrawIndirectCount-format-10390
For each vertex attribute accessed by this command, if its `VkVertexInputAttributeDescription::format` or `VkVertexInputAttributeDescription2EXT::format` is not a **packed format**, the value of `attribAddress`, calculated as described in **Vertex Input Calculation**, **must** be a multiple of the **component size of the format**
- VUID-vkCmdDrawIndirectCount-None-07842
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then `vkCmdSetPrimitiveTopologyEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndirectCount-dynamicPrimitiveTopologyUnrestricted-07500
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then the `primitiveTopology` parameter of `vkCmdSetPrimitiveTopologyEXT` **must** be of the same **topology class** as the pipeline `VkPipelineInputAssemblyStateCreateInfo::topology` state
- VUID-vkCmdDrawIndirectCount-primitiveTopology-10286
If a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage is bound, then the **current value** of `primitiveTopology` **must** be `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST` prior to this drawing command
- VUID-vkCmdDrawIndirectCount-primitiveTopology-10747
If `vkCmdSetPrimitiveTopologyEXT` set `primitiveTopology` to `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST` prior to this drawing command, then a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage **must** be bound
- VUID-vkCmdDrawIndirectCount-primitiveTopology-10748
If `vkCmdSetPrimitiveTopologyEXT` set `primitiveTopology` to `VK_PRIMITIVE_TOPOLOGY_POINT_LIST` prior to this drawing command, both a `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` and `VK_SHADER_STAGE_GEOMETRY_BIT` stage are not bound, then the **Vertex Execution Model** **must** have a **PointSize** decorated variable that is statically written to
- VUID-vkCmdDrawIndirectCount-pStrides-04913
If the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE` dynamic state enabled, but without the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, then `vkCmdBindVertexBuffers2EXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this draw command, and the `pStrides` parameter of `vkCmdBindVertexBuffers2EXT` **must** not be `NULL`
- VUID-vkCmdDrawIndirectCount-None-04914
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then `vkCmdSetVertexInputEXT` **must** have been called and not subsequently **invalidated** in the current command buffer

prior to this draw command

- VUID-vkCmdDrawIndirectCount-Input-07939

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then all variables with the `Input` storage class decorated with `Location` in the `Vertex Execution Model OpEntryPoint` **must** contain a location in `VkVertexInputAttributeDescription2EXT::location`

- VUID-vkCmdDrawIndirectCount-Input-08734

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then the numeric type associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be the same as `VkVertexInputAttributeDescription2EXT::format`

- VUID-vkCmdDrawIndirectCount-format-08936

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and `VkVertexInputAttributeDescription2EXT::format` has a 64-bit component, then the scalar width associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be 64-bit

- VUID-vkCmdDrawIndirectCount-format-08937

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and the scalar width associated with a `Location` decorated `Input` variable in the `Vertex Execution Model OpEntryPoint` is 64-bit, then the corresponding `VkVertexInputAttributeDescription2EXT::format` **must** have a 64-bit component

- VUID-vkCmdDrawIndirectCount-None-09203

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and `VkVertexInputAttributeDescription2EXT::format` has a 64-bit component, then all `Input` variables at the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** not use components that are not present in the format

- VUID-vkCmdDrawIndirectCount-None-04875

If the bound graphics pipeline state was created with both a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage and the `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT` dynamic state enabled, and the `current value` of `primitiveTopology` is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, then `vkCmdSetPatchControlPointsEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-04879

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled then `vkCmdSetPrimitiveRestartEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndirectCount-None-09637

If the `input assembly` is `VK_PRIMITIVE_TOPOLOGY_POINT_LIST`, `VK_PRIMITIVE_TOPOLOGY_LINE_LIST`, `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST`, `VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY`, or

`VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY`, the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled, then `vkCmdSetPrimitiveRestartEnableEXT` **must** be `VK_FALSE`

- VUID-vkCmdDrawIndirectCount-None-10909

If the `input assembly` is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled then `vkCmdSetPrimitiveRestartEnableEXT` **must** be `VK_FALSE`

- VUID-vkCmdDrawIndirectCount-buffer-02708

If `buffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdDrawIndirectCount-buffer-02709

`buffer` **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set

- VUID-vkCmdDrawIndirectCount-offset-02710

`offset` **must** be a multiple of 4

- VUID-vkCmdDrawIndirectCount-commandBuffer-02711

`commandBuffer` **must** not be a protected command buffer

- VUID-vkCmdDrawIndirectCount-countBuffer-02714

If `countBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdDrawIndirectCount-countBuffer-02715

`countBuffer` **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set

- VUID-vkCmdDrawIndirectCount-countBufferOffset-02716

`countBufferOffset` **must** be a multiple of 4

- VUID-vkCmdDrawIndirectCount-countBuffer-02717

The count stored in `countBuffer` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDrawIndirectCount`

- VUID-vkCmdDrawIndirectCount-countBufferOffset-04129

$(\text{countBufferOffset} + \text{sizeof}(\text{uint32\_t}))$ **must** be less than or equal to the size of `countBuffer`

- VUID-vkCmdDrawIndirectCount-None-04445

If the `drawIndirectCount` feature is not enabled this function **must** not be used

- VUID-vkCmdDrawIndirectCount-stride-03110

`stride` **must** be a multiple of 4 and **must** be greater than or equal to `sizeof(VkDrawIndirectCommand)`

- VUID-vkCmdDrawIndirectCount-maxDrawCount-03111

If `maxDrawCount` is greater than or equal to 1, $(\text{stride} \times (\text{maxDrawCount} - 1) + \text{offset} + \text{sizeof}(\text{VkDrawIndirectCommand}))$ **must** be less than or equal to the size of `buffer`

Valid Usage (Implicit)

- VUID-vkCmdDrawIndirectCount-commandBuffer-parameter

`commandBuffer` **must** be a valid `VkCommandBuffer` handle

- VUID-vkCmdDrawIndirectCount-buffer-parameter
buffer **must** be a valid [VkBuffer](#) handle
- VUID-vkCmdDrawIndirectCount-countBuffer-parameter
countBuffer **must** be a valid [VkBuffer](#) handle
- VUID-vkCmdDrawIndirectCount-commandBuffer-recording
commandBuffer **must** be in the [recording state](#)
- VUID-vkCmdDrawIndirectCount-commandBuffer-cmdpool
The [VkCommandPool](#) that **commandBuffer** was allocated from **must** support [VK_QUEUE_GRAPHICS_BIT](#) operations
- VUID-vkCmdDrawIndirectCount-renderpass
This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawIndirectCount-commonparent
Each of **buffer**, **commandBuffer**, and **countBuffer** **must** have been created, allocated, or retrieved from the same [VkDevice](#)

Host Synchronization

- Host access to **commandBuffer** **must** be externally synchronized
- Host access to the [VkCommandPool](#) that **commandBuffer** was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Inside	VK_QUEUE_GRAPHICS_BIT	Action

To record an indexed indirect drawing command, call:

```
// Provided by VK_VERSION_1_0
void vkCmdDrawIndexedIndirect(
    VkCommandBuffer          commandBuffer,
    VkBuffer                 buffer,
    VkDeviceSize             offset,
    uint32_t                 drawCount,
    uint32_t                 stride);
```

- **commandBuffer** is the command buffer into which the command is recorded.
- **buffer** is the buffer containing draw parameters.
- **offset** is the byte offset into **buffer** where parameters begin.

- `drawCount` is the number of draws to execute, and **can** be zero.
- `stride` is the byte stride between successive sets of draw parameters.

`vkCmdDrawIndexedIndirect` behaves similarly to `vkCmdDrawIndexed` except that the parameters are read by the device from a buffer during execution. `drawCount` draws are executed by the command, with parameters taken from `buffer` starting at `offset` and increasing by `stride` bytes for each successive draw. The parameters of each draw are encoded in an array of `VkDrawIndexedIndirectCommand` structures. If `drawCount` is less than or equal to one, `stride` is ignored.

Valid Usage

- VUID-vkCmdDrawIndexedIndirect-magFilter-04553
If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdDrawIndexedIndirect-magFilter-09598
If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- VUID-vkCmdDrawIndexedIndirect-mipmapMode-04770
If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdDrawIndexedIndirect-mipmapMode-09599
If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- VUID-vkCmdDrawIndexedIndirect-unnormalizedCoordinates-09635
If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `levelCount` and `layerCount` **must** be 1
- VUID-vkCmdDrawIndexedIndirect-None-08609
If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `viewType` **must** be `VK_IMAGE_VIEW_TYPE_1D` or `VK_IMAGE_VIEW_TYPE_2D`
- VUID-vkCmdDrawIndexedIndirect-None-08610

If a [VkSampler](#) created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a [VkImageView](#) as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name

- VUID-vkCmdDrawIndexedIndirect-None-08611

If a [VkSampler](#) created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a [VkImageView](#) as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values

- VUID-vkCmdDrawIndexedIndirect-aspectMask-06478

If a [VkImageView](#) is sampled with `depth comparison`, the image view **must** have been created with an `aspectMask` that contains `VK_IMAGE_ASPECT_DEPTH_BIT`

- VUID-vkCmdDrawIndexedIndirect-None-02691

If a [VkImageView](#) is accessed using atomic operations as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT`

- VUID-vkCmdDrawIndexedIndirect-None-07888

If a `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's `format` `features` **must** contain `VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT`

- VUID-vkCmdDrawIndexedIndirect-None-02692

If a [VkImageView](#) is sampled with `VK_FILTER_CUBIC_EXT` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT`

- VUID-vkCmdDrawIndexedIndirect-None-02693

If the `VK_EXT_filter_cubic` extension is not enabled and any [VkImageView](#) is sampled with `VK_FILTER_CUBIC_EXT` as a result of this command, it **must** not have a `VkImageViewType` of `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`

- VUID-vkCmdDrawIndexedIndirect-filterCubic-02694

Any [VkImageView](#) being sampled with `VK_FILTER_CUBIC_EXT` as a result of this command **must** have a `VkImageViewType` and format that supports cubic filtering, as specified by `VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic` returned by `vkGetPhysicalDeviceImageFormatProperties2`

- VUID-vkCmdDrawIndexedIndirect-filterCubicMinmax-02695

Any [VkImageView](#) being sampled with `VK_FILTER_CUBIC_EXT` with a reduction mode of either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` as a result of this command **must** have a `VkImageViewType` and format that supports cubic filtering together with minmax filtering, as specified by `VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax` returned by `vkGetPhysicalDeviceImageFormatProperties2`

- VUID-vkCmdDrawIndexedIndirect-None-08600

For each set n that is statically used by a `bound shader`, a descriptor set **must** have been bound to n at the same pipeline bind point, with a `VkPipelineLayout` that is compatible for set n , with the `VkPipelineLayout` used to create the current `VkPipeline`, as described in

Pipeline Layout Compatibility

- VUID-vkCmdDrawIndexedIndirect-None-08601

For each push constant that is statically used by a [bound shader](#), a push constant value **must** have been set for the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for push constants, with the [VkPipelineLayout](#) used to create the current [VkPipeline](#), as described in [Pipeline Layout Compatibility](#)

- VUID-vkCmdDrawIndexedIndirect-None-10068

For each array of resources that is used by a [bound shader](#), the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command

- VUID-vkCmdDrawIndexedIndirect-None-08114

Descriptors in each bound descriptor set, specified via [vkCmdBindDescriptorSets](#), **must** be valid as described by [descriptor validity](#) if they are statically used by a [bound shader](#)

- VUID-vkCmdDrawIndexedIndirect-imageLayout-00344

If an image descriptor is accessed by a shader, the [VkImageLayout](#) **must** match the subresource accessible from the [VkImageView](#) as defined by the [image layout matching rules](#)

- VUID-vkCmdDrawIndexedIndirect-None-08606

A valid pipeline **must** be bound to the pipeline bind point used by this command

- VUID-vkCmdDrawIndexedIndirect-None-08608

There **must** not have been any calls to dynamic state setting commands for any state specified statically in the [VkPipeline](#) object bound to the pipeline bind point used by this command, since that pipeline was bound

- VUID-vkCmdDrawIndexedIndirect-uniformBuffers-06935

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a uniform buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndexedIndirect-storageBuffers-06936

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a storage buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndexedIndirect-commandBuffer-02707

If [commandBuffer](#) is an unprotected command buffer and [protectedNoFault](#) is not supported, any resource accessed by [bound shaders](#) **must** not be a protected resource

- VUID-vkCmdDrawIndexedIndirect-viewType-07752

If a [VkImageView](#) is accessed as a result of this command, then the image view's [viewType](#) **must** match the [Dim](#) operand of the [OpTypeImage](#) as described in [Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types](#)

- VUID-vkCmdDrawIndexedIndirect-format-07753

If a [VkImageView](#) or [VkBufferView](#) is accessed as a result of this command, then the [numeric type](#) of the view's [format](#) and the [Sampled Type](#) operand of the [OpTypeImage](#) **must**

match

- VUID-vkCmdDrawIndexedIndirect-OpImageWrite-08795

If a `VkImageView` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the image view's format

- VUID-vkCmdDrawIndexedIndirect-OpImageWrite-04469

If a `VkBufferView` is accessed using `OpImageWrite` as a result of this command, then the `Type` of the `Texel` operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDrawIndexedIndirect-SampledType-04470

If a `VkImageView` with a `VkFormat` that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64

- VUID-vkCmdDrawIndexedIndirect-SampledType-04471

If a `VkImageView` with a `VkFormat` that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32

- VUID-vkCmdDrawIndexedIndirect-SampledType-04472

If a `VkBufferView` with a `VkFormat` that has a 64-bit component width is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 64

- VUID-vkCmdDrawIndexedIndirect-SampledType-04473

If a `VkBufferView` with a `VkFormat` that has a component width less than 64-bit is accessed as a result of this command, the `SampledType` of the `OpTypeImage` operand of that instruction **must** have a `Width` of 32

- VUID-vkCmdDrawIndexedIndirect-sparseImageInt64Atomics-04474

If the `sparseImageInt64Atomics` feature is not enabled, `VkImage` objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command

- VUID-vkCmdDrawIndexedIndirect-sparseImageInt64Atomics-04475

If the `sparseImageInt64Atomics` feature is not enabled, `VkBuffer` objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command

- VUID-vkCmdDrawIndexedIndirect-None-07288

Any shader invocation executed by this command **must** `terminate`

- VUID-vkCmdDrawIndexedIndirect-None-09600

If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, all image subresources identified by that descriptor **must** be in the image layout identified when the descriptor was written

- VUID-vkCmdDrawIndexedIndirect-renderPass-02684

The current render pass **must** be `compatible` with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to

VK_PIPELINE_BIND_POINT_GRAPHICS

- VUID-vkCmdDrawIndexedIndirect-subpass-02685

The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`

- VUID-vkCmdDrawIndexedIndirect-None-07748

If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set

- VUID-vkCmdDrawIndexedIndirect-OpTypeImage-07468

If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass

- VUID-vkCmdDrawIndexedIndirect-None-07469

Input attachment views accessed in a subpass **must** be created with the same `VkFormat` as the corresponding subpass definition, and be created with a `VkImageView` that is compatible with the attachment referenced by the subpass' `pInputAttachments` [`InputAttachmentIndex`] in the bound `VkFramebuffer` as specified by [Fragment Input Attachment Compatibility](#)

- VUID-vkCmdDrawIndexedIndirect-None-06537

Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexedIndirect-None-10795

If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexedIndirect-None-10796

If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexedIndirect-None-10797

If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexedIndirect-None-06539

If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment

- VUID-vkCmdDrawIndexedIndirect-None-06886

If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, `depth writes` **must** be disabled

- VUID-vkCmdDrawIndexedIndirect-None-06887
If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back **writeMask** are not zero, and stencil test is enabled, **all stencil ops must** be **VK_STENCIL_OP_KEEP**
- VUID-vkCmdDrawIndexedIndirect-None-07831
If the bound graphics pipeline state was created with the **VK_DYNAMIC_STATE_VIEWPORT** dynamic state enabled then **vkCmdSetViewport must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07832
If the bound graphics pipeline state was created with the **VK_DYNAMIC_STATE_SCISSOR** dynamic state enabled then **vkCmdSetScissor must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-08617
If a graphics pipeline is bound which was created with the **VK_DYNAMIC_STATE_LINE_WIDTH** dynamic state enabled, the **current value** of **rasterizerDiscardEnable** is **VK_FALSE**, and the **effective rasterization input topology** is in line topology class, then **vkCmdSetLineWidth must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07834
If a graphics pipeline is bound which was created with the **VK_DYNAMIC_STATE_DEPTH_BIAS** dynamic state enabled, the **current value** of **rasterizerDiscardEnable** is **VK_FALSE**, and the **current value** of **depthBiasEnable** is **VK_TRUE**, then **vkCmdSetDepthBias must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07835
If a graphics pipeline is bound which was created with the **VK_DYNAMIC_STATE_BLEND_CONSTANTS** dynamic state enabled, the **current value** of **rasterizerDiscardEnable** is **VK_FALSE**, and an active color attachment **current value** of **blendEnable** is **VK_TRUE** with a blend equations where any **VkBlendFactor** member is **VK_BLEND_FACTOR_CONSTANT_COLOR**, **VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR**, **VK_BLEND_FACTOR_CONSTANT_ALPHA**, or **VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA**, then **vkCmdSetBlendConstants must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07836
If a graphics pipeline is bound which was created with the **VK_DYNAMIC_STATE_DEPTH_BOUNDS** dynamic state enabled, the **current value** of **rasterizerDiscardEnable** is **VK_FALSE**, and the **current value** of **depthBoundsTestEnable** is **VK_TRUE**, then **vkCmdSetDepthBounds must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirect-None-07837
If a graphics pipeline is bound which was created with the **VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK** dynamic state enabled, the **current value** of **rasterizerDiscardEnable** is **VK_FALSE**, and the **current value** of **stencilTestEnable** is **VK_TRUE**, then **vkCmdSetStencilCompareMask must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07838

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilWriteMask` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07839

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_REFERENCE` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilReference` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-maxMultiviewInstanceIndex-02688

If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to `VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex`

- VUID-vkCmdDrawIndexedIndirect-sampleLocationsEnable-02689

If the bound graphics pipeline was created with `VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable` set to `VK_TRUE`, then the active depth attachment **must** have been created with the `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` bit set

- VUID-vkCmdDrawIndexedIndirect-None-06666

If the `VK_EXT_sample_locations` extension is enabled, a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `sampleLocationsEnable` is `VK_TRUE`, then `vkCmdSetSampleLocationsEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07840

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_CULL_MODE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetCullModeEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07841

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_FRONT_FACE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetFrontFaceEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07843

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, `vkCmdSetDepthTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07844

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthTestEnable` is `VK_TRUE`, then `vkCmdSetDepthWriteEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07845

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthTestEnable` is `VK_TRUE`, then `vkCmdSetDepthCompareOpEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07846

If the `depthBounds` feature is enabled, a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetDepthBoundsTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07847

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetStencilTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-07848

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_OP` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilOpEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-viewportCount-03417

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, then `vkCmdSetViewportWithCountEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-scissorCount-03418

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` dynamic state enabled, then `vkCmdSetScissorWithCountEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing

- VUID-vkCmdDrawIndexedIndirect-viewportCount-03419

If a graphics pipeline is bound which was created with both the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic states enabled, then the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** match the `scissorCount` parameter of `vkCmdSetScissorWithCountEXT`

- VUID-vkCmdDrawIndexedIndirect-None-04876

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` dynamic state enabled, then `vkCmdSetRasterizerDiscardEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-04877

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetDepthBiasEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-logicOp-04878

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LOGIC_OP_EXT` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `logicOpEnable` is `VK_TRUE`, then `vkCmdSetLogicOpEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-primitiveFragmentShadingRateWithMultipleViewports-04552

If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the `PrimitiveShadingRateKHR` built-in, then `vkCmdSetViewportWithCountEXT` **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be 1

- VUID-vkCmdDrawIndexedIndirect-blendEnable-04727

If a graphics pipeline is bound which was created with `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then for each color attachment, if the corresponding image view's `format` `features` do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the corresponding `current value` of `blendEnable` **must** be `VK_FALSE`

- VUID-vkCmdDrawIndexedIndirect-None-08644

If a graphics pipeline is bound, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then the `current value` of `rasterizationSamples` **must** be the same as the current color and/or depth/stencil attachments

- VUID-vkCmdDrawIndexedIndirect-maxFragmentDualSrcAttachments-09239

If `blending` is enabled for any attachment where either the source or destination blend factors for that attachment `use the secondary color input`, the maximum value of `Location` for any output attachment `statically used` in the `Fragment Execution Model` executed by this command **must** be less than `maxFragmentDualSrcAttachments`

- VUID-vkCmdDrawIndexedIndirect-None-04007

All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or `VK_NULL_HANDLE` buffers

bound

- VUID-vkCmdDrawIndexedIndirect-None-04008

If the `nullDescriptor` feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be `VK_NULL_HANDLE`

- VUID-vkCmdDrawIndexedIndirect-None-02721

If the `robustBufferAccess` feature is not enabled, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in [Vertex Input Description](#)

- VUID-vkCmdDrawIndexedIndirect-format-10389

For each vertex attribute accessed by this command, if its `VkVertexInputAttributeDescription::format` or `VkVertexInputAttributeDescription2EXT::format` is a `packed format`, the value of `attribAddress`, calculated as described in [Vertex Input Calculation](#), **must** be a multiple of the `size of the format`

- VUID-vkCmdDrawIndexedIndirect-format-10390

For each vertex attribute accessed by this command, if its `VkVertexInputAttributeDescription::format` or `VkVertexInputAttributeDescription2EXT::format` is not a `packed format`, the value of `attribAddress`, calculated as described in [Vertex Input Calculation](#), **must** be a multiple of the `component size of the format`

- VUID-vkCmdDrawIndexedIndirect-None-07842

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then `vkCmdSetPrimitiveTopologyEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-dynamicPrimitiveTopologyUnrestricted-07500

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then the `primitiveTopology` parameter of `vkCmdSetPrimitiveTopologyEXT` **must** be of the same `topology class` as the pipeline `VkPipelineInputAssemblyStateCreateInfo::topology` state

- VUID-vkCmdDrawIndexedIndirect-primitiveTopology-10286

If a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage is bound, then the `current value` of `primitiveTopology` **must** be `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST` prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-primitiveTopology-10747

If `vkCmdSetPrimitiveTopologyEXT` set `primitiveTopology` to `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST` prior to this drawing command, then a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage **must** be bound

- VUID-vkCmdDrawIndexedIndirect-primitiveTopology-10748

If `vkCmdSetPrimitiveTopologyEXT` set `primitiveTopology` to `VK_PRIMITIVE_TOPOLOGY_POINT_LIST` prior to this drawing command, both a `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` and `VK_SHADER_STAGE_GEOMETRY_BIT` stage are not bound, then the `Vertex Execution Model` **must** have a `PointSize` decorated variable that is statically written to

- VUID-vkCmdDrawIndexedIndirect-pStrides-04913

If the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE` dynamic state enabled, but without the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, then `vkCmdBindVertexBuffers2EXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this draw command, and the `pStrides` parameter of `vkCmdBindVertexBuffers2EXT` **must** not be `NULL`

- VUID-vkCmdDrawIndexedIndirect-None-04914

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then `vkCmdSetVertexInputEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this draw command

- VUID-vkCmdDrawIndexedIndirect-Input-07939

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then all variables with the `Input` storage class decorated with `Location` in the `Vertex Execution Model OpEntryPoint` **must** contain a location in `VkVertexInputAttributeDescription2EXT::location`

- VUID-vkCmdDrawIndexedIndirect-Input-08734

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then the numeric type associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be the same as `VkVertexInputAttributeDescription2EXT::format`

- VUID-vkCmdDrawIndexedIndirect-format-08936

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and `VkVertexInputAttributeDescription2EXT::format` has a 64-bit component, then the scalar width associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be 64-bit

- VUID-vkCmdDrawIndexedIndirect-format-08937

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and the scalar width associated with a `Location` decorated `Input` variable in the `Vertex Execution Model OpEntryPoint` is 64-bit, then the corresponding `VkVertexInputAttributeDescription2EXT::format` **must** have a 64-bit component

- VUID-vkCmdDrawIndexedIndirect-None-09203

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and `VkVertexInputAttributeDescription2EXT::format` has a 64-bit component, then all `Input` variables at the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** not use components that are not present in the format

- VUID-vkCmdDrawIndexedIndirect-None-04875

If the bound graphics pipeline state was created with both a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage and the `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT` dynamic state enabled, and the `current value` of `primitiveTopology` is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, then `vkCmdSetPatchControlPointsEXT` **must** have been called and not subsequently

invalidated in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-04879

If the bound graphics pipeline state was created with the **VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE** dynamic state enabled then **vkCmdSetPrimitiveRestartEnableEXT** **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirect-None-09637

If the **input assembly** is **VK_PRIMITIVE_TOPOLOGY_POINT_LIST**, **VK_PRIMITIVE_TOPOLOGY_LINE_LIST**, **VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST**, **VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY**, or **VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY**, the bound graphics pipeline state was created with the **VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE** dynamic state enabled, then **vkCmdSetPrimitiveRestartEnableEXT** **must** be **VK_FALSE**

- VUID-vkCmdDrawIndexedIndirect-None-10909

If the **input assembly** is **VK_PRIMITIVE_TOPOLOGY_PATCH_LIST**, the bound graphics pipeline state was created with the **VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE** dynamic state enabled then **vkCmdSetPrimitiveRestartEnableEXT** **must** be **VK_FALSE**

- VUID-vkCmdDrawIndexedIndirect-buffer-02708

If **buffer** is non-sparse then it **must** be bound completely and contiguously to a single **VkDeviceMemory** object

- VUID-vkCmdDrawIndexedIndirect-buffer-02709

buffer **must** have been created with the **VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT** bit set

- VUID-vkCmdDrawIndexedIndirect-offset-02710

offset **must** be a multiple of 4

- VUID-vkCmdDrawIndexedIndirect-commandBuffer-02711

commandBuffer **must** not be a protected command buffer

- VUID-vkCmdDrawIndexedIndirect-drawCount-02718

If the **multiDrawIndirect** feature is not enabled, **drawCount** **must** be 0 or 1

- VUID-vkCmdDrawIndexedIndirect-drawCount-02719

drawCount **must** be less than or equal to **VkPhysicalDeviceLimits::maxDrawIndirectCount**

- VUID-vkCmdDrawIndexedIndirect-None-07312

A valid index buffer **must** be bound

- VUID-vkCmdDrawIndexedIndirect-drawCount-00528

If **drawCount** is greater than 1, **stride** **must** be a multiple of 4 and **must** be greater than or equal to **sizeof(VkDrawIndexedIndirectCommand)**

- VUID-vkCmdDrawIndexedIndirect-drawCount-00539

If **drawCount** is equal to 1, (**offset** + **sizeof(VkDrawIndexedIndirectCommand)**) **must** be less than or equal to the size of **buffer**

- VUID-vkCmdDrawIndexedIndirect-drawCount-00540

If **drawCount** is greater than 1, (**stride** × (**drawCount** - 1) + **offset** + **sizeof(VkDrawIndexedIndirectCommand)**) **must** be less than or equal to the size of **buffer**

Valid Usage (Implicit)

- VUID-vkCmdDrawIndexedIndirect-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdDrawIndexedIndirect-buffer-parameter
`buffer` **must** be a valid `VkBuffer` handle
- VUID-vkCmdDrawIndexedIndirect-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdDrawIndexedIndirect-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdDrawIndexedIndirect-renderpass
This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawIndexedIndirect-commonparent
Both of `buffer`, and `commandBuffer` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Inside	VK_QUEUE_GRAPHICS_BIT	Action

The `VkDrawIndexedIndirectCommand` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDrawIndexedIndirectCommand {
    uint32_t    indexCount;
    uint32_t    instanceCount;
    uint32_t    firstIndex;
    int32_t     vertexOffset;
    uint32_t    firstInstance;
} VkDrawIndexedIndirectCommand;
```

- `indexCount` is the number of vertices to draw.
- `instanceCount` is the number of instances to draw.
- `firstIndex` is the base index within the index buffer.
- `vertexOffset` is the value added to the vertex index before indexing into the vertex buffer.
- `firstInstance` is the instance ID of the first instance to draw.

The members of `VkDrawIndexedIndirectCommand` have the same meaning as the similarly named parameters of `vkCmdDrawIndexed`.

Valid Usage

- VUID-VkDrawIndexedIndirectCommand-pNext-09461

If the bound graphics pipeline state was created with `VkPipelineVertexInputDivisorStateCreateInfo` in the `pNext` chain of `VkGraphicsPipelineCreateInfo::pVertexInputState`, any member of `VkPipelineVertexInputDivisorStateCreateInfo::pVertexBindingDivisors` has a value other than `1` in `divisor`, and `VkPhysicalDeviceVertexAttributeDivisorProperties::supportsNonZeroFirstInstance` is `VK_FALSE`, then `firstInstance` **must** be `0`

- VUID-VkDrawIndexedIndirectCommand-None-09462

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, any member of the `pVertexBindingDescriptions` parameter to the `vkCmdSetVertexInputEXT` call that sets this dynamic state has a value other than `1` in `divisor`, and `VkPhysicalDeviceVertexAttributeDivisorProperties::supportsNonZeroFirstInstance` is `VK_FALSE`, then `firstInstance` **must** be `0`

- VUID-VkDrawIndexedIndirectCommand-robustBufferAccess2-08798

If the `robustBufferAccess2` feature is not enabled, $(\text{indexSize} \times (\text{firstIndex} + \text{indexCount}) + \text{offset})$ **must** be less than or equal to the size of the bound index buffer, with `indexSize` being based on the type specified by `indexType`, where the index buffer, `indexType`, and `offset` are specified via `vkCmdBindIndexBuffer`

- VUID-VkDrawIndexedIndirectCommand-firstInstance-00554

If the `drawIndirectFirstInstance` feature is not enabled, `firstInstance` **must** be `0`

To record an indexed draw call with a draw call count sourced from a buffer, call:

```
// Provided by VK_VERSION_1_2
void vkCmdDrawIndexedIndirectCount(
    VkCommandBuffer          commandBuffer,
    VkBuffer                 buffer,
    VkDeviceSize             offset,
    VkBuffer                 countBuffer,
    VkDeviceSize             countBufferOffset,
    uint32_t                 maxDrawCount,
    uint32_t                 stride);
```

- **commandBuffer** is the command buffer into which the command is recorded.
- **buffer** is the buffer containing draw parameters.
- **offset** is the byte offset into **buffer** where parameters begin.
- **countBuffer** is the buffer containing the draw count.
- **countBufferOffset** is the byte offset into **countBuffer** where the draw count begins.
- **maxDrawCount** specifies the maximum number of draws that will be executed. The actual number of executed draw calls is the minimum of the count specified in **countBuffer** and **maxDrawCount**.
- **stride** is the byte stride between successive sets of draw parameters.

vkCmdDrawIndexedIndirectCount behaves similarly to **vkCmdDrawIndexedIndirect** except that the draw count is read by the device from a buffer during execution. The command will read an unsigned 32-bit integer from **countBuffer** located at **countBufferOffset** and use this as the draw count.

Valid Usage

- VUID-vkCmdDrawIndexedIndirectCount-magFilter-04553
If a **VkSampler** created with **magFilter** or **minFilter** equal to **VK_FILTER_LINEAR**, **reductionMode** equal to **VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE**, and **compareEnable** equal to **VK_FALSE** is used to sample a **VkImageView** as a result of this command, then the image view's **format features** **must** contain **VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT**
- VUID-vkCmdDrawIndexedIndirectCount-magFilter-09598
If a **VkSampler** created with **magFilter** or **minFilter** equal to **VK_FILTER_LINEAR** and **reductionMode** equal to either **VK_SAMPLER_REDUCTION_MODE_MIN** or **VK_SAMPLER_REDUCTION_MODE_MAX** is used to sample a **VkImageView** as a result of this command, then the image view's **format features** **must** contain **VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT**
- VUID-vkCmdDrawIndexedIndirectCount-mipmapMode-04770
If a **VkSampler** created with **mipmapMode** equal to **VK_SAMPLER_MIPMAP_MODE_LINEAR**, **reductionMode** equal to **VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE**, and **compareEnable** equal to **VK_FALSE** is used to sample a **VkImageView** as a result of this command, then the image view's **format features** **must** contain **VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT**

- VUID-vkCmdDrawIndexedIndirectCount-mipmapMode-09599
If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- VUID-vkCmdDrawIndexedIndirectCount-unnormalizedCoordinates-09635
If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `levelCount` and `layerCount` **must** be 1
- VUID-vkCmdDrawIndexedIndirectCount-None-08609
If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `viewType` **must** be `VK_IMAGE_VIEW_TYPE_1D` or `VK_IMAGE_VIEW_TYPE_2D`
- VUID-vkCmdDrawIndexedIndirectCount-None-08610
If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name
- VUID-vkCmdDrawIndexedIndirectCount-None-08611
If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values
- VUID-vkCmdDrawIndexedIndirectCount-aspectMask-06478
If a `VkImageView` is sampled with `depth comparison`, the image view **must** have been created with an `aspectMask` that contains `VK_IMAGE_ASPECT_DEPTH_BIT`
- VUID-vkCmdDrawIndexedIndirectCount-None-02691
If a `VkImageView` is accessed using atomic operations as a result of this command, then the image view's `format features` **must** contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT`
- VUID-vkCmdDrawIndexedIndirectCount-None-07888
If a `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's `format features` **must** contain `VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT`
- VUID-vkCmdDrawIndexedIndirectCount-None-02692
If a `VkImageView` is sampled with `VK_FILTER_CUBIC_EXT` as a result of this command, then the image view's `format features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT`
- VUID-vkCmdDrawIndexedIndirectCount-None-02693
If the `VK_EXT_filter_cubic` extension is not enabled and any `VkImageView` is sampled with `VK_FILTER_CUBIC_EXT` as a result of this command, it **must** not have a `VkImageViewType` of `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`
- VUID-vkCmdDrawIndexedIndirectCount-filterCubic-02694

Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)

- VUID-vkCmdDrawIndexedIndirectCount-filterCubicMinmax-02695

Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) with a reduction mode of either [VK_SAMPLER_REDUCTION_MODE_MIN](#) or [VK_SAMPLER_REDUCTION_MODE_MAX](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering together with minmax filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)

- VUID-vkCmdDrawIndexedIndirectCount-None-08600

For each set n that is statically used by a [bound shader](#), a descriptor set **must** have been bound to n at the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for set n , with the [VkPipelineLayout](#) used to create the current [VkPipeline](#), as described in [Pipeline Layout Compatibility](#)

- VUID-vkCmdDrawIndexedIndirectCount-None-08601

For each push constant that is statically used by a [bound shader](#), a push constant value **must** have been set for the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for push constants, with the [VkPipelineLayout](#) used to create the current [VkPipeline](#), as described in [Pipeline Layout Compatibility](#)

- VUID-vkCmdDrawIndexedIndirectCount-None-10068

For each array of resources that is used by a [bound shader](#), the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command

- VUID-vkCmdDrawIndexedIndirectCount-None-08114

Descriptors in each bound descriptor set, specified via [vkCmdBindDescriptorSets](#), **must** be valid as described by [descriptor validity](#) if they are statically used by a [bound shader](#)

- VUID-vkCmdDrawIndexedIndirectCount-imageLayout-00344

If an image descriptor is accessed by a shader, the [VkImageLayout](#) **must** match the subresource accessible from the [VkImageView](#) as defined by the [image layout matching rules](#)

- VUID-vkCmdDrawIndexedIndirectCount-None-08606

A valid pipeline **must** be bound to the pipeline bind point used by this command

- VUID-vkCmdDrawIndexedIndirectCount-None-08608

There **must** not have been any calls to dynamic state setting commands for any state specified statically in the [VkPipeline](#) object bound to the pipeline bind point used by this command, since that pipeline was bound

- VUID-vkCmdDrawIndexedIndirectCount-uniformBuffers-06935

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a uniform buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndexedIndirectCount-storageBuffers-06936

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a storage buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDrawIndexedIndirectCount-commandBuffer-02707

If [commandBuffer](#) is an unprotected command buffer and [protectedNoFault](#) is not supported, any resource accessed by [bound shaders](#) **must** not be a protected resource

- VUID-vkCmdDrawIndexedIndirectCount-viewType-07752

If a [VkImageView](#) is accessed as a result of this command, then the image view's [viewType](#) **must** match the [Dim](#) operand of the [OpTypeImage](#) as described in [Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types](#)

- VUID-vkCmdDrawIndexedIndirectCount-format-07753

If a [VkImageView](#) or [VkBufferView](#) is accessed as a result of this command, then the [numeric type](#) of the view's [format](#) and the [Sampled Type](#) operand of the [OpTypeImage](#) **must** match

- VUID-vkCmdDrawIndexedIndirectCount-OpImageWrite-08795

If a [VkImageView](#) is accessed using [OpImageWrite](#) as a result of this command, then the [Type](#) of the [Texel](#) operand of that instruction **must** have at least as many components as the image view's format

- VUID-vkCmdDrawIndexedIndirectCount-OpImageWrite-04469

If a [VkBufferView](#) is accessed using [OpImageWrite](#) as a result of this command, then the [Type](#) of the [Texel](#) operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDrawIndexedIndirectCount-SampledType-04470

If a [VkImageView](#) with a [VkFormat](#) that has a 64-bit component width is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 64

- VUID-vkCmdDrawIndexedIndirectCount-SampledType-04471

If a [VkImageView](#) with a [VkFormat](#) that has a component width less than 64-bit is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 32

- VUID-vkCmdDrawIndexedIndirectCount-SampledType-04472

If a [VkBufferView](#) with a [VkFormat](#) that has a 64-bit component width is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 64

- VUID-vkCmdDrawIndexedIndirectCount-SampledType-04473

If a [VkBufferView](#) with a [VkFormat](#) that has a component width less than 64-bit is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 32

- VUID-vkCmdDrawIndexedIndirectCount-sparseImageInt64Atomics-04474

If the [sparseImageInt64Atomics](#) feature is not enabled, [VkImage](#) objects created with the [VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT](#) flag **must** not be accessed by atomic instructions through an [OpTypeImage](#) with a [SampledType](#) with a [Width](#) of 64 by this command

- VUID-vkCmdDrawIndexedIndirectCount-sparseImageInt64Atomics-04475
If the `sparseImageInt64Atomics` feature is not enabled, `VkBuffer` objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDrawIndexedIndirectCount-None-07288
Any shader invocation executed by this command **must** `terminate`
- VUID-vkCmdDrawIndexedIndirectCount-None-09600
If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, all image subresources identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDrawIndexedIndirectCount-renderPass-02684
The current render pass **must** be `compatible` with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawIndexedIndirectCount-subpass-02685
The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`
- VUID-vkCmdDrawIndexedIndirectCount-None-07748
If any shader statically accesses an input attachment, a valid descriptor **must** be bound to the pipeline via a descriptor set
- VUID-vkCmdDrawIndexedIndirectCount-OpTypeImage-07468
If any shader executed by this pipeline accesses an `OpTypeImage` variable with a `Dim` operand of `SubpassData`, it **must** be decorated with an `InputAttachmentIndex` that corresponds to a valid input attachment in the current subpass
- VUID-vkCmdDrawIndexedIndirectCount-None-07469
Input attachment views accessed in a subpass **must** be created with the same `VkFormat` as the corresponding subpass definition, and be created with a `VkImageView` that is compatible with the attachment referenced by the subpass' `pInputAttachments` [`InputAttachmentIndex`] in the bound `VkFramebuffer` as specified by `Fragment Input Attachment Compatibility`
- VUID-vkCmdDrawIndexedIndirectCount-None-06537
Memory backing image subresources used as attachments in the current render pass **must** not be written in any way other than as an attachment by this command
- VUID-vkCmdDrawIndexedIndirectCount-None-10795
If a color attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command
- VUID-vkCmdDrawIndexedIndirectCount-None-10796
If a depth attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexedIndirectCount-None-10797

If a stencil attachment is written by any prior command in this subpass or by the load, store, or resolve operations for this subpass,

it **must** not be accessed in any way other than as an attachment by this command

- VUID-vkCmdDrawIndexedIndirectCount-None-06539

If any previously recorded command in the current subpass accessed an image subresource used as an attachment in this subpass in any way other than as an attachment, this command **must** not write to that image subresource as an attachment

- VUID-vkCmdDrawIndexedIndirectCount-None-06886

If the current render pass instance uses a depth/stencil attachment with a read-only layout for the depth aspect, **depth writes must** be disabled

- VUID-vkCmdDrawIndexedIndirectCount-None-06887

If the current render pass instance uses a depth/stencil attachment with a read-only layout for the stencil aspect, both front and back **writeMask** are not zero, and stencil test is enabled, **all stencil ops must** be **VK_STENCIL_OP_KEEP**

- VUID-vkCmdDrawIndexedIndirectCount-None-07831

If the bound graphics pipeline state was created with the **VK_DYNAMIC_STATE_VIEWPORT** dynamic state enabled then **vkCmdSetViewport must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07832

If the bound graphics pipeline state was created with the **VK_DYNAMIC_STATE_SCISSOR** dynamic state enabled then **vkCmdSetScissor must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-08617

If a graphics pipeline is bound which was created with the **VK_DYNAMIC_STATE_LINE_WIDTH** dynamic state enabled, the **current value** of **rasterizerDiscardEnable** is **VK_FALSE**, and the **effective rasterization input topology** is in line topology class, then **vkCmdSetLineWidth must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07834

If a graphics pipeline is bound which was created with the **VK_DYNAMIC_STATE_DEPTH_BIAS** dynamic state enabled, the **current value** of **rasterizerDiscardEnable** is **VK_FALSE**, and the **current value** of **depthBiasEnable** is **VK_TRUE**, then **vkCmdSetDepthBias must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07835

If a graphics pipeline is bound which was created with the **VK_DYNAMIC_STATE_BLEND_CONSTANTS** dynamic state enabled, the **current value** of **rasterizerDiscardEnable** is **VK_FALSE**, and an active color attachment **current value** of **blendEnable** is **VK_TRUE** with a blend equations where any **VkBlendFactor** member is **VK_BLEND_FACTOR_CONSTANT_COLOR**, **VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR**, **VK_BLEND_FACTOR_CONSTANT_ALPHA**, or **VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA**, then

`vkCmdSetBlendConstants` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07836

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BOUNDS` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthBoundsTestEnable` is `VK_TRUE`, then `vkCmdSetDepthBounds` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07837

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilCompareMask` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07838

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilWriteMask` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07839

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_REFERENCE` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilReference` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-maxMultiviewInstanceIndex-02688

If the draw is recorded in a render pass instance with multiview enabled, the maximum instance index **must** be less than or equal to `VkPhysicalDeviceMultiviewProperties::maxMultiviewInstanceIndex`

- VUID-vkCmdDrawIndexedIndirectCount-sampleLocationsEnable-02689

If the bound graphics pipeline was created with `VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable` set to `VK_TRUE`, then the active depth attachment **must** have been created with the `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` bit set

- VUID-vkCmdDrawIndexedIndirectCount-None-06666

If the `VK_EXT_sample_locations` extension is enabled, a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `sampleLocationsEnable` is `VK_TRUE`, then `vkCmdSetSampleLocationsEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07840

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_CULL_MODE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then

`vkCmdSetCullModeEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07841

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_FRONT_FACE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetFrontFaceEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07843

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, `vkCmdSetDepthTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07844

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthTestEnable` is `VK_TRUE`, then `vkCmdSetDepthWriteEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07845

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, and the `current value` of `depthTestEnable` is `VK_TRUE`, then `vkCmdSetDepthCompareOpEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07846

If the `depthBounds` feature is enabled, a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetDepthBoundsTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07847

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE` dynamic state enabled, and the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetStencilTestEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-07848

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_STENCIL_OP` dynamic state enabled, the `current value` of `rasterizerDiscardEnable` is `VK_FALSE`, the `current value` of `stencilTestEnable` is `VK_TRUE`, then `vkCmdSetStencilOpEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-viewportCount-03417

If a graphics pipeline is bound which was created with the

`VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, then `vkCmdSetViewportWithCountEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-scissorCount-03418

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` dynamic state enabled, then `vkCmdSetScissorWithCountEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing

- VUID-vkCmdDrawIndexedIndirectCount-viewportCount-03419

If a graphics pipeline is bound which was created with both the `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` and `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic states enabled, then the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** match the `scissorCount` parameter of `vkCmdSetScissorWithCountEXT`

- VUID-vkCmdDrawIndexedIndirectCount-None-04876

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE` dynamic state enabled, then `vkCmdSetRasterizerDiscardEnableEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-04877

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE` dynamic state enabled, and the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, then `vkCmdSetDepthBiasEnableEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-logicOp-04878

If a graphics pipeline is bound which was created with the `VK_DYNAMIC_STATE_LOGIC_OP_EXT` dynamic state enabled, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, and the **current value** of `logicOpEnable` is `VK_TRUE`, then `vkCmdSetLogicOpEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-primitiveFragmentShadingRateWithMultipleViewports-04552

If the `primitiveFragmentShadingRateWithMultipleViewports` limit is not supported, the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` dynamic state enabled, and any of the shader stages of the bound graphics pipeline write to the `PrimitiveShadingRateKHR` built-in, then `vkCmdSetViewportWithCountEXT` **must** have been called in the current command buffer prior to this drawing command, and the `viewportCount` parameter of `vkCmdSetViewportWithCountEXT` **must** be 1

- VUID-vkCmdDrawIndexedIndirectCount-blendEnable-04727

If a graphics pipeline is bound which was created with `VK_DYNAMIC_STATE_COLOR_BLEND_ENABLE_EXT` dynamic state enabled, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, then for each color attachment, if the corresponding image view's **format features** do not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`, then the corresponding **current value** of `blendEnable` **must** be `VK_FALSE`

- VUID-vkCmdDrawIndexedIndirectCount-None-08644
If a graphics pipeline is bound, the **current value** of `rasterizerDiscardEnable` is `VK_FALSE`, then the **current value** of `rasterizationSamples` **must** be the same as the current color and/or depth/stencil attachments
- VUID-vkCmdDrawIndexedIndirectCount-maxFragmentDualSrcAttachments-09239
If **blending** is enabled for any attachment where either the source or destination blend factors for that attachment **use the secondary color input**, the maximum value of `Location` for any output attachment **statically used** in the `Fragment Execution Model` executed by this command **must** be less than `maxFragmentDualSrcAttachments`
- VUID-vkCmdDrawIndexedIndirectCount-None-04007
All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have either valid or `VK_NULL_HANDLE` buffers bound
- VUID-vkCmdDrawIndexedIndirectCount-None-04008
If the `nullDescriptor` feature is not enabled, all vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** not be `VK_NULL_HANDLE`
- VUID-vkCmdDrawIndexedIndirectCount-None-02721
If the `robustBufferAccess` feature is not enabled, then for a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in [Vertex Input Description](#)
- VUID-vkCmdDrawIndexedIndirectCount-format-10389
For each vertex attribute accessed by this command, if its `VkVertexInputAttributeDescription::format` or `VkVertexInputAttributeDescription2EXT::format` is a **packed format**, the value of `attribAddress`, calculated as described in [Vertex Input Calculation](#), **must** be a multiple of the **size of the format**
- VUID-vkCmdDrawIndexedIndirectCount-format-10390
For each vertex attribute accessed by this command, if its `VkVertexInputAttributeDescription::format` or `VkVertexInputAttributeDescription2EXT::format` is not a **packed format**, the value of `attribAddress`, calculated as described in [Vertex Input Calculation](#), **must** be a multiple of the **component size of the format**
- VUID-vkCmdDrawIndexedIndirectCount-None-07842
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then `vkCmdSetPrimitiveTopologyEXT` **must** have been called and not subsequently **invalidated** in the current command buffer prior to this drawing command
- VUID-vkCmdDrawIndexedIndirectCount-dynamicPrimitiveTopologyUnrestricted-07500
If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY` dynamic state enabled then the `primitiveTopology` parameter of `vkCmdSetPrimitiveTopologyEXT` **must** be of the same **topology class** as the pipeline `VkPipelineInputAssemblyStateCreateInfo::topology` state
- VUID-vkCmdDrawIndexedIndirectCount-primitiveTopology-10286
If a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage is bound, then the **current value** of

`primitiveTopology` **must** be `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST` prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-primitiveTopology-10747

If `vkCmdSetPrimitiveTopologyEXT` set `primitiveTopology` to `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST` prior to this drawing command, then a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage **must** be bound

- VUID-vkCmdDrawIndexedIndirectCount-primitiveTopology-10748

If `vkCmdSetPrimitiveTopologyEXT` set `primitiveTopology` to `VK_PRIMITIVE_TOPOLOGY_POINT_LIST` prior to this drawing command, both a `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT` and `VK_SHADER_STAGE_GEOMETRY_BIT` stage are not bound, then the `Vertex Execution Model` **must** have a `PointSize` decorated variable that is statically written to

- VUID-vkCmdDrawIndexedIndirectCount-pStrides-04913

If the bound graphics pipeline was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE` dynamic state enabled, but without the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled, then `vkCmdBindVertexBuffers2EXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this draw command, and the `pStrides` parameter of `vkCmdBindVertexBuffers2EXT` **must** not be `NULL`

- VUID-vkCmdDrawIndexedIndirectCount-None-04914

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then `vkCmdSetVertexInputEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this draw command

- VUID-vkCmdDrawIndexedIndirectCount-Input-07939

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then all variables with the `Input` storage class decorated with `Location` in the `Vertex Execution Model OpEntryPoint` **must** contain a location in `VkVertexInputAttributeDescription2EXT::location`

- VUID-vkCmdDrawIndexedIndirectCount-Input-08734

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then the numeric type associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be the same as `VkVertexInputAttributeDescription2EXT::format`

- VUID-vkCmdDrawIndexedIndirectCount-format-08936

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and `VkVertexInputAttributeDescription2EXT::format` has a 64-bit component, then the scalar width associated with all `Input` variables of the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** be 64-bit

- VUID-vkCmdDrawIndexedIndirectCount-format-08937

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and the scalar width associated with a `Location` decorated `Input` variable in the `Vertex Execution Model OpEntryPoint` is 64-bit, then the corresponding `VkVertexInputAttributeDescription2EXT::format` **must** have a

64-bit component

- VUID-vkCmdDrawIndexedIndirectCount-None-09203

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled and `VkVertexInputAttributeDescription2EXT::format` has a 64-bit component, then all `Input` variables at the corresponding `Location` in the `Vertex Execution Model OpEntryPoint` **must** not use components that are not present in the format

- VUID-vkCmdDrawIndexedIndirectCount-None-04875

If the bound graphics pipeline state was created with both a `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT` stage and the `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT` dynamic state enabled, and the `current value` of `primitiveTopology` is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, then `vkCmdSetPatchControlPointsEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-04879

If the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled then `vkCmdSetPrimitiveRestartEnableEXT` **must** have been called and not subsequently `invalidated` in the current command buffer prior to this drawing command

- VUID-vkCmdDrawIndexedIndirectCount-None-09637

If the `input assembly` is `VK_PRIMITIVE_TOPOLOGY_POINT_LIST`, `VK_PRIMITIVE_TOPOLOGY_LINE_LIST`, `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST`, `VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY`, or `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY`, the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled, then `vkCmdSetPrimitiveRestartEnableEXT` **must** be `VK_FALSE`

- VUID-vkCmdDrawIndexedIndirectCount-None-10909

If the `input assembly` is `VK_PRIMITIVE_TOPOLOGY_PATCH_LIST`, the bound graphics pipeline state was created with the `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE` dynamic state enabled then `vkCmdSetPrimitiveRestartEnableEXT` **must** be `VK_FALSE`

- VUID-vkCmdDrawIndexedIndirectCount-buffer-02708

If `buffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdDrawIndexedIndirectCount-buffer-02709

`buffer` **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set

- VUID-vkCmdDrawIndexedIndirectCount-offset-02710

`offset` **must** be a multiple of 4

- VUID-vkCmdDrawIndexedIndirectCount-commandBuffer-02711

`commandBuffer` **must** not be a protected command buffer

- VUID-vkCmdDrawIndexedIndirectCount-countBuffer-02714

If `countBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

- VUID-vkCmdDrawIndexedIndirectCount-countBuffer-02715

`countBuffer` **must** have been created with the `VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT` bit set

- VUID-vkCmdDrawIndexedIndirectCount-countBufferOffset-02716
`countBufferOffset` **must** be a multiple of 4
- VUID-vkCmdDrawIndexedIndirectCount-countBuffer-02717
The count stored in `countBuffer` **must** be less than or equal to `VkPhysicalDeviceLimits::maxDrawIndirectCount`
- VUID-vkCmdDrawIndexedIndirectCount-countBufferOffset-04129
(`countBufferOffset + sizeof(uint32_t)`) **must** be less than or equal to the size of `countBuffer`
- VUID-vkCmdDrawIndexedIndirectCount-None-04445
If the `drawIndirectCount` feature is not enabled this function **must** not be used
- VUID-vkCmdDrawIndexedIndirectCount-None-07312
A valid index buffer **must** be bound
- VUID-vkCmdDrawIndexedIndirectCount-stride-03142
`stride` **must** be a multiple of 4 and **must** be greater than or equal to `sizeof(VkDrawIndexedIndirectCommand)`
- VUID-vkCmdDrawIndexedIndirectCount-maxDrawCount-03143
If `maxDrawCount` is greater than or equal to 1, (`stride × (maxDrawCount - 1) + offset + sizeof(VkDrawIndexedIndirectCommand)`) **must** be less than or equal to the size of `buffer`

Valid Usage (Implicit)

- VUID-vkCmdDrawIndexedIndirectCount-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdDrawIndexedIndirectCount-buffer-parameter
`buffer` **must** be a valid `VkBuffer` handle
- VUID-vkCmdDrawIndexedIndirectCount-countBuffer-parameter
`countBuffer` **must** be a valid `VkBuffer` handle
- VUID-vkCmdDrawIndexedIndirectCount-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdDrawIndexedIndirectCount-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdDrawIndexedIndirectCount-renderpass
This command **must** only be called inside of a render pass instance
- VUID-vkCmdDrawIndexedIndirectCount-commonparent
Each of `buffer`, `commandBuffer`, and `countBuffer` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Inside	VK_QUEUE_GRAPHIC S_BIT	Action

Chapter 22. Fixed-Function Vertex Processing

Vertex fetching is controlled via configurable state, as a logically distinct graphics pipeline stage.

22.1. Vertex Attributes

Vertex shaders **can** define input variables, which receive *vertex attribute* data transferred from one or more `VkBuffer`(s) by drawing commands. Vertex shader input variables are bound to buffers via an indirect binding where the vertex shader associates a *vertex input attribute* number with each variable, vertex input attributes are associated to *vertex input bindings* on a per-pipeline basis, and vertex input bindings are associated with specific buffers on a per-draw basis via the `vkCmdBindVertexBuffers` command. Vertex input attribute and vertex input binding descriptions also contain format information controlling how data is extracted from buffer memory and converted to the format expected by the vertex shader.

There are `VkPhysicalDeviceLimits::maxVertexInputAttributes` number of vertex input attributes and `VkPhysicalDeviceLimits::maxVertexInputBindings` number of vertex input bindings (each referred to by zero-based indices), where there are at least as many vertex input attributes as there are vertex input bindings. Applications **can** store multiple vertex input attributes interleaved in a single buffer, and use a single vertex input binding to access those attributes.

In GLSL, vertex shaders associate input variables with a vertex input attribute number using the `location` layout qualifier. The `Component` layout qualifier associates components of a vertex shader input variable with components of a vertex input attribute.

GLSL Example

```
// Assign location M to variableName
layout (location=M, component=2) in vec2 variableName;

// Assign locations [N,N+L) to the array elements of variableNameArray
layout (location=N) in vec4 variableNameArray[L];
```

In SPIR-V, vertex shaders associate input variables with a vertex input attribute number using the `Location` decoration. The `Component` decoration associates components of a vertex shader input variable with components of a vertex input attribute. The `Location` and `Component` decorations are specified via the `OpDecorate` instruction.

SPIR-V Example

```
...
%1 = OpExtInstImport "GLSL.std.450"
...
OpName %9 "variableName"
OpName %15 "variableNameArray"
OpDecorate %18 BuiltIn VertexIndex
```

```

OpDecorate %19 BuiltIn InstanceIndex
OpDecorate %9 Location M
OpDecorate %9 Component 2
OpDecorate %15 Location N
...
%2 = OpTypeVoid
%3 = OpTypeFunction %2
%6 = OpTypeFloat 32
%7 = OpTypeVector %6 2
%8 = OpTypePointer Input %7
%9 = OpVariable %8 Input
%10 = OpTypeVector %6 4
%11 = OpTypeInt 32 0
%12 = OpConstant %11 L
%13 = OpTypeArray %10 %12
%14 = OpTypePointer Input %13
%15 = OpVariable %14 Input
...

```

22.1.1. Attribute Location and Component Assignment

The **Location** decoration specifies which vertex input attribute is used to read and interpret the data that a variable will consume.

When a vertex shader input variable declared using a 16- or 32-bit scalar or vector data type is assigned a **Location**, its value(s) are taken from the components of the input attribute specified with the corresponding `VkVertexInputAttributeDescription::location`. The components used depend on the type of variable and the **Component** decoration specified in the variable declaration, as identified in [Input Attribute Components Accessed By 16-Bit and 32-Bit Input Variables](#). Any 16-bit or 32-bit scalar or vector input will consume a single **Location**. For 16-bit and 32-bit data types, missing components are filled in with default values as described [below](#).

If an implementation supports `storageInputOutput16`, vertex shader input variables **can** have a width of 16 bits.

Table 26. Input Attribute Components Accessed By 16-Bit and 32-Bit Input Variables

16-bit or 32-bit data type	Component decoration	Components consumed
scalar	0 or unspecified	(x, o, o, o)
scalar	1	(o, y, o, o)
scalar	2	(o, o, z, o)
scalar	3	(o, o, o, w)
two-component vector	0 or unspecified	(x, y, o, o)
two-component vector	1	(o, y, z, o)
two-component vector	2	(o, o, z, w)

16-bit or 32-bit data type	Component decoration	Components consumed
three-component vector	0 or unspecified	(x, y, z, o)
three-component vector	1	(o, y, z, w)
four-component vector	0 or unspecified	(x, y, z, w)

Components indicated by “o” are available for use by other input variables which are sourced from the same attribute, and if used, are either filled with the corresponding component from the input format (if present), or the default value.

When a vertex shader input variable declared using a 32-bit floating-point matrix type is assigned a **Location** *i*, its values are taken from consecutive input attributes starting with the corresponding `VkVertexInputAttributeDescription::location`. Such matrices are treated as an array of column vectors with values taken from the input attributes identified in [Input Attributes Accessed by 32-Bit Input Matrix Variables](#). The `VkVertexInputAttributeDescription::format` **must** be specified with a `VkFormat` that corresponds to the appropriate type of column vector. The **Component** decoration **must** not be used with matrix types.

Table 27. Input Attributes Accessed by 32-Bit Input Matrix Variables

Data type	Column vector type	Locations consumed	Components consumed
mat2	two-component vector	i, i+1	(x, y, o, o), (x, y, o, o)
mat2x3	three-component vector	i, i+1	(x, y, z, o), (x, y, z, o)
mat2x4	four-component vector	i, i+1	(x, y, z, w), (x, y, z, w)
mat3x2	two-component vector	i, i+1, i+2	(x, y, o, o), (x, y, o, o), (x, y, o, o)
mat3	three-component vector	i, i+1, i+2	(x, y, z, o), (x, y, z, o), (x, y, z, o)
mat3x4	four-component vector	i, i+1, i+2	(x, y, z, w), (x, y, z, w), (x, y, z, w)
mat4x2	two-component vector	i, i+1, i+2, i+3	(x, y, o, o), (x, y, o, o), (x, y, o, o), (x, y, o, o)
mat4x3	three-component vector	i, i+1, i+2, i+3	(x, y, z, o), (x, y, z, o), (x, y, z, o), (x, y, z, o)
mat4	four-component vector	i, i+1, i+2, i+3	(x, y, z, w), (x, y, z, w), (x, y, z, w), (x, y, z, w)

Components indicated by “o” are available for use by other input variables which are sourced from the same attribute, and if used, are either filled with the corresponding component from the input (if present), or the default value.

When a vertex shader input variable declared using a scalar or vector 64-bit data type is assigned a **Location** *i*, its values are taken from consecutive input attributes starting with the corresponding

VkVertexInputAttributeDescription::location. The **Location** slots and **Component** words used depend on the type of variable and the **Component** decoration specified in the variable declaration, as identified in [Input Attribute Locations and Components Accessed by 64-Bit Input Variables](#). For 64-bit data types, no default attribute values are provided. Input variables **must** not use more components than provided by the attribute.

Table 28. Input Attribute Locations and Components Accessed by 64-Bit Input Variables

Input format	Locations consumed	64-bit data type	Location decoration	Component decoration	32-bit component s consumed
R64	i	scalar	i	0 or unspecified	(x, y, -, -)
R64G64	i	scalar	i	0 or unspecified	(x, y, o, o)
		scalar	i	2	(o, o, z, w)
		two-component vector	i	0 or unspecified	(x, y, z, w)
R64G64B64	i, i+1	scalar	i	0 or unspecified	(x, y, o, o), (o, o, -, -)
		scalar	i	2	(o, o, z, w), (o, o, -, -)
		scalar	i+1	0 or unspecified	(o, o, o, o), (x, y, -, -)
		two-component vector	i	0 or unspecified	(x, y, z, w), (o, o, -, -)
		three-component vector	i	unspecified	(x, y, z, w), (x, y, -, -)
R64G64B64A64	i, i+1	scalar	i	0 or unspecified	(x, y, o, o), (o, o, o, o)
		scalar	i	2	(o, o, z, w), (o, o, o, o)
		scalar	i+1	0 or unspecified	(o, o, o, o), (x, y, o, o)
		scalar	i+1	2	(o, o, o, o), (o, o, z, w)
		two-component vector	i	0 or unspecified	(x, y, z, w), (o, o, o, o)
		two-component vector	i+1	0 or unspecified	(o, o, o, o), (x, y, z, w)
		three-component vector	i	unspecified	(x, y, z, w), (x, y, o, o)
		four-component vector	i	unspecified	(x, y, z, w), (x, y, z, w)

Components indicated by “o” are available for use by other input variables which are sourced from the same attribute. Components indicated by “-” are not available for input variables as there are no default values provided for 64-bit data types, and there is no data provided by the input format.

When a vertex shader input variable declared using a 64-bit floating-point matrix type is assigned a **Location** *i*, its values are taken from consecutive input attribute locations. Such matrices are treated as an array of column vectors with values taken from the input attributes as shown in [Input Attribute Locations and Components Accessed by 64-Bit Input Variables](#). Each column vector starts at the **Location** immediately following the last **Location** of the previous column vector. The number of attributes and components assigned to each matrix is determined by the matrix dimensions and ranges from two to eight locations.

When a vertex shader input variable declared using an array type is assigned a location, its values are taken from consecutive input attributes starting with the corresponding **VkVertexInputAttributeDescription::location**. The number of attributes and components assigned to each element are determined according to the data type of the array elements and **Component** decoration (if any) specified in the declaration of the array, as described above. Each element of the array, in order, is assigned to consecutive locations, but all at the same specified component within each location.

Only input variables declared with the data types and component decorations as specified above are supported. Two variables are allowed to share the same **Location** slot only if their **Component** words do not overlap. If multiple variables share the same **Location** slot, they **must** all have the same SPIR-V floating-point component type or all have the same width scalar type components.

22.2. Vertex Input Description

Applications specify vertex input attribute and vertex input binding descriptions as part of graphics pipeline creation by setting the **VkGraphicsPipelineCreateInfo::pVertexInputState** pointer to a **VkPipelineVertexInputStateCreateInfo** structure. Alternatively, if the graphics pipeline is created with the **VK_DYNAMIC_STATE_VERTEX_INPUT_EXT** dynamic state enabled, then the vertex input attribute and vertex input binding descriptions are specified dynamically with **vkCmdSetVertexInputEXT**, and the **VkGraphicsPipelineCreateInfo::pVertexInputState** pointer is ignored.

The **VkPipelineVertexInputStateCreateInfo** structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineVertexInputStateCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkPipelineVertexInputStateCreateFlags flags;
    uint32_t                  vertexBindingDescriptionCount;
    const VkVertexInputBindingDescription* pVertexBindingDescriptions;
    uint32_t                  vertexAttributeDescriptionCount;
    const VkVertexInputAttributeDescription* pVertexAttributeDescriptions;
} VkPipelineVertexInputStateCreateInfo;
```

- **sType** is a **VkStructureType** value identifying this structure.

- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `vertexBindingDescriptionCount` is the number of vertex binding descriptions provided in `pVertexBindingDescriptions`.
- `pVertexBindingDescriptions` is a pointer to an array of `VkVertexInputBindingDescription` structures.
- `vertexAttributeDescriptionCount` is the number of vertex attribute descriptions provided in `pVertexAttributeDescriptions`.
- `pVertexAttributeDescriptions` is a pointer to an array of `VkVertexInputAttributeDescription` structures.

Valid Usage

- VUID-VkPipelineVertexInputStateCreateInfo-vertexBindingDescriptionCount-00613
`vertexBindingDescriptionCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputBindings`
- VUID-VkPipelineVertexInputStateCreateInfo-vertexAttributeDescriptionCount-00614
`vertexAttributeDescriptionCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputAttributes`
- VUID-VkPipelineVertexInputStateCreateInfo-binding-00615
For every `binding` specified by each element of `pVertexAttributeDescriptions`, a `VkVertexInputBindingDescription` **must** exist in `pVertexBindingDescriptions` with the same value of `binding`
- VUID-VkPipelineVertexInputStateCreateInfo-pVertexBindingDescriptions-00616
All elements of `pVertexBindingDescriptions` **must** describe distinct binding numbers
- VUID-VkPipelineVertexInputStateCreateInfo-pVertexAttributeDescriptions-00617
All elements of `pVertexAttributeDescriptions` **must** describe distinct attribute locations

Valid Usage (Implicit)

- VUID-VkPipelineVertexInputStateCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO`
- VUID-VkPipelineVertexInputStateCreateInfo-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkPipelineVertexInputDivisorStateCreateInfo`
- VUID-VkPipelineVertexInputStateCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkPipelineVertexInputStateCreateInfo-flags-zero bitmask
`flags` **must** be 0
- VUID-VkPipelineVertexInputStateCreateInfo-pVertexBindingDescriptions-parameter
If `vertexBindingDescriptionCount` is not 0, `pVertexBindingDescriptions` **must** be a valid

pointer to an array of `vertexBindingDescriptionCount` valid `VkVertexInputBindingDescription` structures

- VUID-VkPipelineVertexInputStateCreateInfo-pVertexAttributeDescriptions-parameter
If `vertexAttributeDescriptionCount` is not 0, `pVertexAttributeDescriptions` **must** be a valid pointer to an array of `vertexAttributeDescriptionCount` valid `VkVertexInputAttributeDescription` structures

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineVertexInputStateCreateFlags;
```

`VkPipelineVertexInputStateCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

Each vertex input binding is specified by the `VkVertexInputBindingDescription` structure, defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkVertexInputBindingDescription {
    uint32_t      binding;
    uint32_t      stride;
    VkVertexInputRate inputRate;
} VkVertexInputBindingDescription;
```

- `binding` is the binding number that this structure describes.
- `stride` is the byte stride between consecutive elements within the buffer.
- `inputRate` is a `VkVertexInputRate` value specifying whether vertex attribute addressing is a function of the vertex index or of the instance index.

Valid Usage

- VUID-VkVertexInputBindingDescription-binding-00618
`binding` **must** be less than `VkPhysicalDeviceLimits::maxVertexInputBindings`
- VUID-VkVertexInputBindingDescription-stride-00619
`stride` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputBindingStride`

Valid Usage (Implicit)

- VUID-VkVertexInputBindingDescription-inputRate-parameter
`inputRate` **must** be a valid `VkVertexInputRate` value

Possible values of `VkVertexInputBindingDescription::inputRate`, specifying the rate at which vertex attributes are pulled from buffers, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkVertexInputRate {
    VK_VERTEX_INPUT_RATE_VERTEX = 0,
    VK_VERTEX_INPUT_RATE_INSTANCE = 1,
} VkVertexInputRate;
```

- `VK_VERTEX_INPUT_RATE_VERTEX` specifies that vertex attribute addressing is a function of the vertex index.
- `VK_VERTEX_INPUT_RATE_INSTANCE` specifies that vertex attribute addressing is a function of the instance index.

Each vertex input attribute is specified by the `VkVertexInputAttributeDescription` structure, defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkVertexInputAttributeDescription {
    uint32_t    location;
    uint32_t    binding;
    VkFormat     format;
    uint32_t    offset;
} VkVertexInputAttributeDescription;
```

- `location` is the shader input location number for this attribute.
- `binding` is the binding number which this attribute takes its data from.
- `format` is the size and type of the vertex attribute data.
- `offset` is a byte offset of this attribute relative to the start of an element in the vertex input binding.

Valid Usage

- VUID-VkVertexInputAttributeDescription-location-00620
`location` **must** be less than `VkPhysicalDeviceLimits::maxVertexInputAttributes`
- VUID-VkVertexInputAttributeDescription-binding-00621
`binding` **must** be less than `VkPhysicalDeviceLimits::maxVertexInputBindings`
- VUID-VkVertexInputAttributeDescription-offset-00622
`offset` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputAttributeOffset`
- VUID-VkVertexInputAttributeDescription-format-00623
The `format features` of `format` **must** contain `VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT`

Valid Usage (Implicit)

- VUID-VkVertexInputAttributeDescription-format-parameter **format** must be a valid [VkFormat](#) value

To [dynamically set](#) the vertex input attribute and vertex input binding descriptions, call:

```
// Provided by VK_EXT_vertex_input_dynamic_state
void vkCmdSetVertexInputEXT(
    VkCommandBuffer                commandBuffer,
    uint32_t                       vertexBindingDescriptionCount,
    const VkVertexInputBindingDescription2EXT* pVertexBindingDescriptions,
    uint32_t                       vertexAttributeDescriptionCount,
    const VkVertexInputAttributeDescription2EXT* pVertexAttributeDescriptions);
```

- **commandBuffer** is the command buffer into which the command will be recorded.
- **vertexBindingDescriptionCount** is the number of vertex binding descriptions provided in **pVertexBindingDescriptions**.
- **pVertexBindingDescriptions** is a pointer to an array of [VkVertexInputBindingDescription2EXT](#) structures.
- **vertexAttributeDescriptionCount** is the number of vertex attribute descriptions provided in **pVertexAttributeDescriptions**.
- **pVertexAttributeDescriptions** is a pointer to an array of [VkVertexInputAttributeDescription2EXT](#) structures.

This command sets the vertex input attribute and vertex input binding descriptions state for subsequent drawing commands when the graphics pipeline is created with [VK_DYNAMIC_STATE_VERTEX_INPUT_EXT](#) set in [VkPipelineDynamicStateCreateInfo::pDynamicStates](#). Otherwise, this state is specified by the [VkGraphicsPipelineCreateInfo::pVertexInputState](#) values used to create the currently active pipeline.

If the bound pipeline state object was also created with the [VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE](#) dynamic state enabled, then [vkCmdBindVertexBuffers2EXT](#) can be used instead of [vkCmdSetVertexInputEXT](#) to dynamically set the stride.

The vertex attribute description for any location in the range `[0,VkPhysicalDeviceLimits::maxVertexInputAttributes)` not specified in the **pVertexAttributeDescriptions** array becomes undefined.

Valid Usage

- VUID-vkCmdSetVertexInputEXT-None-04790
The **vertexInputDynamicState** feature **must** be enabled
- VUID-vkCmdSetVertexInputEXT-vertexBindingDescriptionCount-04791

`vertexBindingDescriptionCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputBindings`

- VUID-vkCmdSetVertexInputEXT-vertexAttributeDescriptionCount-04792
`vertexAttributeDescriptionCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputAttributes`
- VUID-vkCmdSetVertexInputEXT-binding-04793
For every `binding` specified by each element of `pVertexAttributeDescriptions`, a `VkVertexInputBindingDescription2EXT` **must** exist in `pVertexBindingDescriptions` with the same value of `binding`
- VUID-vkCmdSetVertexInputEXT-pVertexBindingDescriptions-04794
All elements of `pVertexBindingDescriptions` **must** describe distinct binding numbers
- VUID-vkCmdSetVertexInputEXT-pVertexAttributeDescriptions-04795
All elements of `pVertexAttributeDescriptions` **must** describe distinct attribute locations

Valid Usage (Implicit)

- VUID-vkCmdSetVertexInputEXT-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetVertexInputEXT-pVertexBindingDescriptions-parameter
If `vertexBindingDescriptionCount` is not 0, `pVertexBindingDescriptions` **must** be a valid pointer to an array of `vertexBindingDescriptionCount` valid `VkVertexInputBindingDescription2EXT` structures
- VUID-vkCmdSetVertexInputEXT-pVertexAttributeDescriptions-parameter
If `vertexAttributeDescriptionCount` is not 0, `pVertexAttributeDescriptions` **must** be a valid pointer to an array of `vertexAttributeDescriptionCount` valid `VkVertexInputAttributeDescription2EXT` structures
- VUID-vkCmdSetVertexInputEXT-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetVertexInputEXT-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

The `VkVertexInputBindingDescription2EXT` structure is defined as:

```
// Provided by VK_EXT_vertex_input_dynamic_state
typedef struct VkVertexInputBindingDescription2EXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           binding;
    uint32_t           stride;
    VkVertexInputRate   inputRate;
    uint32_t           divisor;
} VkVertexInputBindingDescription2EXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `binding` is the binding number that this structure describes.
- `stride` is the byte stride between consecutive elements within the buffer.
- `inputRate` is a `VkVertexInputRate` value specifying whether vertex attribute addressing is a function of the vertex index or of the instance index.
- `divisor` is the number of successive instances that will use the same value of the vertex attribute when instanced rendering is enabled. This member **can** be a value other than `1` if the `vertexAttributeInstanceRateDivisor` feature is enabled. For example, if the divisor is `N`, the same vertex attribute will be applied to `N` successive instances before moving on to the next vertex attribute. The maximum value of `divisor` is implementation-dependent and can be queried using `VkPhysicalDeviceVertexAttributeDivisorPropertiesEXT::maxVertexAttribDivisor`. A value of `0` **can** be used for the divisor if the `vertexAttributeInstanceRateZeroDivisor` feature is enabled. In this case, the same vertex attribute will be applied to all instances.

Valid Usage

- VUID-VkVertexInputBindingDescription2EXT-binding-04796
`binding` **must** be less than `VkPhysicalDeviceLimits::maxVertexInputBindings`
- VUID-VkVertexInputBindingDescription2EXT-stride-04797
`stride` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputBindingStride`
- VUID-VkVertexInputBindingDescription2EXT-divisor-04798
If the `vertexAttributeInstanceRateZeroDivisor` feature is not enabled, `divisor` **must** not be

0

- VUID-VkVertexInputBindingDescription2EXT-divisor-04799
If the `vertexAttributeInstanceRateDivisor` feature is not enabled, `divisor` **must** be 1
- VUID-VkVertexInputBindingDescription2EXT-divisor-06226
`divisor` **must** be a value between 0 and `VkPhysicalDeviceVertexAttributeDivisorPropertiesEXT::maxVertexAttribDivisor`, inclusive
- VUID-VkVertexInputBindingDescription2EXT-divisor-06227
If `divisor` is not 1 then `inputRate` **must** be of type `VK_VERTEX_INPUT_RATE_INSTANCE`

Valid Usage (Implicit)

- VUID-VkVertexInputBindingDescription2EXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_VERTEX_INPUT_BINDING_DESCRIPTION_2_EXT`
- VUID-VkVertexInputBindingDescription2EXT-inputRate-parameter
`inputRate` **must** be a valid `VkVertexInputRate` value

The `VkVertexInputAttributeDescription2EXT` structure is defined as:

```
// Provided by VK_EXT_vertex_input_dynamic_state
typedef struct VkVertexInputAttributeDescription2EXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           location;
    uint32_t           binding;
    VkFormat            format;
    uint32_t           offset;
} VkVertexInputAttributeDescription2EXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `location` is the shader input location number for this attribute.
- `binding` is the binding number which this attribute takes its data from.
- `format` is the size and type of the vertex attribute data.
- `offset` is a byte offset of this attribute relative to the start of an element in the vertex input binding.

Valid Usage

- VUID-VkVertexInputAttributeDescription2EXT-location-06228
`location` **must** be less than `VkPhysicalDeviceLimits::maxVertexInputAttributes`
- VUID-VkVertexInputAttributeDescription2EXT-binding-06229
`binding` **must** be less than `VkPhysicalDeviceLimits::maxVertexInputBindings`

- VUID-VkVertexInputAttributeDescription2EXT-offset-06230
offset **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputAttributeOffset`
- VUID-VkVertexInputAttributeDescription2EXT-format-04805
The `format` features of `format` **must** contain `VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT`

Valid Usage (Implicit)

- VUID-VkVertexInputAttributeDescription2EXT-sType-sType
sType **must** be `VK_STRUCTURE_TYPE_VERTEX_INPUT_ATTRIBUTE_DESCRIPTION_2_EXT`
- VUID-VkVertexInputAttributeDescription2EXT-format-parameter
format **must** be a valid `VkFormat` value

To bind vertex buffers to a command buffer for use in subsequent drawing commands, call:

```
// Provided by VK_VERSION_1_0
void vkCmdBindVertexBuffers(
    VkCommandBuffer          commandBuffer,
    uint32_t                 firstBinding,
    uint32_t                 bindingCount,
    const VkBuffer*          pBuffer,
    const VkDeviceSize*      pOffsets);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `firstBinding` is the index of the first vertex input binding whose state is updated by the command.
- `bindingCount` is the number of vertex input bindings whose state is updated by the command.
- `pBuffers` is a pointer to an array of buffer handles.
- `pOffsets` is a pointer to an array of buffer offsets.

The values taken from elements `i` of `pBuffers` and `pOffsets` replace the current state for the vertex input binding `firstBinding + i`, for `i` in `[0, bindingCount)`. The vertex input binding is updated to start at the offset indicated by `pOffsets[i]` from the start of the buffer `pBuffers[i]`. All vertex input attributes that use each of these bindings will use these updated addresses in their address calculations for subsequent drawing commands. If the `nullDescriptor` feature is enabled, elements of `pBuffers` **can** be `VK_NULL_HANDLE`, and **can** be used by the vertex shader. If a vertex input attribute is bound to a vertex input binding that is `VK_NULL_HANDLE`, the values taken from memory are considered to be zero, and missing G, B, or A components are filled with (0,0,1).

Valid Usage

- VUID-vkCmdBindVertexBuffers-firstBinding-00624
firstBinding **must** be less than `VkPhysicalDeviceLimits::maxVertexInputBindings`

- VUID-vkCmdBindVertexBuffers-firstBinding-00625
The sum of `firstBinding` and `bindingCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputBindings`
- VUID-vkCmdBindVertexBuffers-pOffsets-00626
All elements of `pOffsets` **must** be less than the size of the corresponding element in `pBuffers`
- VUID-vkCmdBindVertexBuffers-pBuffers-00627
All elements of `pBuffers` **must** have been created with the `VK_BUFFER_USAGE_VERTEX_BUFFER_BIT` flag
- VUID-vkCmdBindVertexBuffers-pBuffers-00628
Each element of `pBuffers` that is non-sparse **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdBindVertexBuffers-pBuffers-04001
If the `nullDescriptor` feature is not enabled, all elements of `pBuffers` **must** not be `VK_NULL_HANDLE`
- VUID-vkCmdBindVertexBuffers-pBuffers-04002
If an element of `pBuffers` is `VK_NULL_HANDLE`, then the corresponding element of `pOffsets` **must** be zero

Valid Usage (Implicit)

- VUID-vkCmdBindVertexBuffers-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdBindVertexBuffers-pBuffers-parameter
`pBuffers` **must** be a valid pointer to an array of `bindingCount` valid or `VK_NULL_HANDLE` `VkBuffer` handles
- VUID-vkCmdBindVertexBuffers-pOffsets-parameter
`pOffsets` **must** be a valid pointer to an array of `bindingCount` `VkDeviceSize` values
- VUID-vkCmdBindVertexBuffers-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdBindVertexBuffers-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdBindVertexBuffers-bindingCount-arraylength
`bindingCount` **must** be greater than 0
- VUID-vkCmdBindVertexBuffers-commonparent
Both of `commandBuffer`, and the elements of `pBuffers` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

Alternatively, to bind vertex buffers, along with their sizes and strides, to a command buffer for use in subsequent drawing commands, call:

```
// Provided by VK_EXT_extended_dynamic_state
void vkCmdBindVertexBuffers2EXT(
    VkCommandBuffer                commandBuffer,
    uint32_t                       firstBinding,
    uint32_t                       bindingCount,
    const VkBuffer*                 pBuffers,
    const VkDeviceSize*             pOffsets,
    const VkDeviceSize*             pSizes,
    const VkDeviceSize*             pStrides);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `firstBinding` is the index of the first vertex input binding whose state is updated by the command.
- `bindingCount` is the number of vertex input bindings whose state is updated by the command.
- `pBuffers` is a pointer to an array of buffer handles.
- `pOffsets` is a pointer to an array of buffer offsets.
- `pSizes` is `NULL` or a pointer to an array of the size in bytes of vertex data bound from `pBuffers`.
- `pStrides` is `NULL` or a pointer to an array of buffer strides.

The values taken from elements `i` of `pBuffers` and `pOffsets` replace the current state for the vertex input binding `firstBinding + i`, for `i` in `[0, bindingCount)`. The vertex input binding is updated to start at the offset indicated by `pOffsets[i]` from the start of the buffer `pBuffers[i]`. If `pSizes` is not `NULL` then `pSizes[i]` specifies the bound size of the vertex buffer starting from the corresponding elements of `pBuffers[i]` plus `pOffsets[i]`. All vertex input attributes that use each of these bindings will use these updated addresses in their address calculations for subsequent drawing commands. If the `nullDescriptor` feature is enabled, elements of `pBuffers` **can** be `VK_NULL_HANDLE`, and **can** be used

by the vertex shader. If a vertex input attribute is bound to a vertex input binding that is `VK_NULL_HANDLE`, the values taken from memory are considered to be zero, and missing G, B, or A components are filled with (0,0,1).

This command also dynamically sets the byte strides between consecutive elements within buffer `pBuffers[i]` to the corresponding `pStrides[i]` value when the graphics pipeline is created with `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, strides are specified by the `VkVertexInputBindingDescription::stride` values used to create the currently active pipeline.

If the bound pipeline state object was also created with the `VK_DYNAMIC_STATE_VERTEX_INPUT_EXT` dynamic state enabled then `vkCmdSetVertexInputEXT` can be used instead of `vkCmdBindVertexBuffers2EXT` to set the stride.

NOTE

Unlike the static state to set the same, `pStrides` must be between 0 and the maximum extent of the attributes in the binding. `vkCmdSetVertexInputEXT` does not have this restriction so can be used if other stride values are desired.

Valid Usage

- VUID-vkCmdBindVertexBuffers2-None-08971

At least one of the following **must** be true:

- the `extendedDynamicState` feature is enabled
- VUID-vkCmdBindVertexBuffers2-firstBinding-03355
`firstBinding` **must** be less than `VkPhysicalDeviceLimits::maxVertexInputBindings`
- VUID-vkCmdBindVertexBuffers2-firstBinding-03356
The sum of `firstBinding` and `bindingCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputBindings`
- VUID-vkCmdBindVertexBuffers2-pOffsets-03357
If `pSizes` is not `NULL`, all elements of `pOffsets` **must** be less than the size of the corresponding element in `pBuffers`
- VUID-vkCmdBindVertexBuffers2-pSizes-03358
If `pSizes` is not `NULL`, all elements of `pOffsets` plus `pSizes` **must** be less than or equal to the size of the corresponding element in `pBuffers`
- VUID-vkCmdBindVertexBuffers2-pBuffers-03359
All elements of `pBuffers` **must** have been created with the `VK_BUFFER_USAGE_VERTEX_BUFFER_BIT` flag
- VUID-vkCmdBindVertexBuffers2-pBuffers-03360
Each element of `pBuffers` that is non-sparse **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- VUID-vkCmdBindVertexBuffers2-pBuffers-04111
If the `nullDescriptor` feature is not enabled, all elements of `pBuffers` **must** not be `VK_NULL_HANDLE`
- VUID-vkCmdBindVertexBuffers2-pBuffers-04112

If an element of `pBuffers` is `VK_NULL_HANDLE`, then the corresponding element of `pOffsets` **must** be zero

- VUID-vkCmdBindVertexBuffers2-pStrides-03362

If `pStrides` is not `NULL` each element of `pStrides` **must** be less than or equal to `VkPhysicalDeviceLimits::maxVertexInputBindingStride`

- VUID-vkCmdBindVertexBuffers2-pStrides-06209

If `pStrides` is not `NULL` each element of `pStrides` **must** be either 0 or greater than or equal to the maximum extent of all vertex input attributes fetched from the corresponding binding, where the extent is calculated as the `VkVertexInputAttributeDescription::offset` plus `VkVertexInputAttributeDescription::format` size

Valid Usage (Implicit)

- VUID-vkCmdBindVertexBuffers2-commandBuffer-parameter

`commandBuffer` **must** be a valid `VkCommandBuffer` handle

- VUID-vkCmdBindVertexBuffers2-pBuffers-parameter

`pBuffers` **must** be a valid pointer to an array of `bindingCount` valid or `VK_NULL_HANDLE` `VkBuffer` handles

- VUID-vkCmdBindVertexBuffers2-pOffsets-parameter

`pOffsets` **must** be a valid pointer to an array of `bindingCount` `VkDeviceSize` values

- VUID-vkCmdBindVertexBuffers2-pSizes-parameter

If `pSizes` is not `NULL`, `pSizes` **must** be a valid pointer to an array of `bindingCount` `VkDeviceSize` values

- VUID-vkCmdBindVertexBuffers2-pStrides-parameter

If `pStrides` is not `NULL`, `pStrides` **must** be a valid pointer to an array of `bindingCount` `VkDeviceSize` values

- VUID-vkCmdBindVertexBuffers2-commandBuffer-recording

`commandBuffer` **must** be in the `recording` state

- VUID-vkCmdBindVertexBuffers2-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

- VUID-vkCmdBindVertexBuffers2-bindingCount-arraylength

If any of `pSizes`, or `pStrides` are not `NULL`, `bindingCount` **must** be greater than 0

- VUID-vkCmdBindVertexBuffers2-commonparent

Both of `commandBuffer`, and the elements of `pBuffers` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized

- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally

synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

22.3. Vertex Attribute Divisor in Instanced Rendering

If the `vertexAttributeInstanceRateDivisor` feature is enabled and the `pNext` chain of `VkPipelineVertexInputStateCreateInfo` includes a `VkPipelineVertexInputDivisorStateCreateInfo` structure, then that structure controls how vertex attributes are assigned to an instance when instanced rendering is enabled.

The `VkPipelineVertexInputDivisorStateCreateInfo` structure is defined as:

```
typedef struct VkPipelineVertexInputDivisorStateCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    uint32_t                 vertexBindingDivisorCount;
    const VkVertexInputBindingDivisorDescription* pVertexBindingDivisors;
} VkPipelineVertexInputDivisorStateCreateInfo;
```

```
// Provided by VK_KHR_vertex_attribute_divisor
typedef VkPipelineVertexInputDivisorStateCreateInfo
VkPipelineVertexInputDivisorStateCreateInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `vertexBindingDivisorCount` is the number of elements in the `pVertexBindingDivisors` array.
- `pVertexBindingDivisors` is a pointer to an array of `VkVertexInputBindingDivisorDescription` structures specifying the divisor value for each binding.

Valid Usage (Implicit)

- VUID-VkPipelineVertexInputDivisorStateCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_DIVISOR_STATE_CREATE_INFO`
- VUID-VkPipelineVertexInputDivisorStateCreateInfo-pVertexBindingDivisors-parameter
`pVertexBindingDivisors` **must** be a valid pointer to an array of `vertexBindingDivisorCount` `VkVertexInputBindingDivisorDescription` structures

- VUID-VkPipelineVertexInputDivisorStateCreateInfo-vertexBindingDivisorCount-arraylength
`vertexBindingDivisorCount` **must** be greater than 0

The individual divisor values per binding are specified using the `VkVertexInputBindingDivisorDescription` structure which is defined as:

```
typedef struct VkVertexInputBindingDivisorDescription {
    uint32_t    binding;
    uint32_t    divisor;
} VkVertexInputBindingDivisorDescription;
```

```
// Provided by VK_KHR_vertex_attribute_divisor
typedef VkVertexInputBindingDivisorDescription
VkVertexInputBindingDivisorDescriptionKHR;
```

- `binding` is the binding number for which the divisor is specified.
- `divisor` is the number of successive instances that will use the same value of the vertex attribute when instanced rendering is enabled. For example, if the divisor is N, the same vertex attribute will be applied to N successive instances before moving on to the next vertex attribute. The maximum value of `divisor` is implementation-dependent and can be queried using `VkPhysicalDeviceVertexAttributeDivisorProperties::maxVertexAttribDivisor`. A value of 0 **can** be used for the divisor if the `vertexAttributeInstanceRateZeroDivisor` feature is enabled. In this case, the same vertex attribute will be applied to all instances.

If this structure is not used to define a divisor value for an attribute, then the divisor has a logical default value of 1.

Valid Usage

- VUID-VkVertexInputBindingDivisorDescription-binding-01869
`binding` **must** be less than `VkPhysicalDeviceLimits::maxVertexInputBindings`
- VUID-VkVertexInputBindingDivisorDescription-vertexAttributeInstanceRateZeroDivisor-02228
 If the `vertexAttributeInstanceRateZeroDivisor` feature is not enabled, `divisor` **must** not be 0
- VUID-VkVertexInputBindingDivisorDescription-vertexAttributeInstanceRateDivisor-02229
 If the `vertexAttributeInstanceRateDivisor` feature is not enabled, `divisor` **must** be 1
- VUID-VkVertexInputBindingDivisorDescription-divisor-01870
`divisor` **must** be a value between 0 and `VkPhysicalDeviceVertexAttributeDivisorProperties::maxVertexAttribDivisor`, inclusive
- VUID-VkVertexInputBindingDivisorDescription-inputRate-01871
`VkVertexInputBindingDescription::inputRate` **must** be of type

22.4. Vertex Input Address Calculation

The address of each attribute for each `vertexIndex` and `instanceIndex` is calculated as follows:

- Let `attribDesc` be the member of `VkPipelineVertexInputStateCreateInfo::pVertexAttributeDescriptions` with `VkVertexInputAttributeDescription::location` equal to the vertex input attribute number.
- Let `bindingDesc` be the member of `VkPipelineVertexInputStateCreateInfo::pVertexBindingDescriptions` with `VkVertexInputAttributeDescription::binding` equal to `attribDesc.binding`.
- Let `vertexIndex` be the index of the vertex within the draw (a value between `firstVertex` and `firstVertex+vertexCount` for `vkCmdDraw`, or a value taken from the index buffer plus `vertexOffset` for `vkCmdDrawIndexed`), and let `instanceIndex` be the instance number of the draw (a value between `firstInstance` and `firstInstance+instanceCount`).
- Let `offset` be an array of offsets into the bound vertex buffers specified during `vkCmdBindVertexBuffers` or `vkCmdBindVertexBuffers2EXT` with `pOffsets`.
- Let `divisor` be the member of `VkPipelineVertexInputDivisorStateCreateInfo::pVertexBindingDivisors` with `VkVertexInputBindingDivisorDescription::binding` equal to `attribDesc.binding`. If the vertex binding state is dynamically set, instead let `divisor` be the member of the `pVertexBindingDescriptions` parameter to the `vkCmdSetVertexInputEXT` call with `VkVertexInputBindingDescription2EXT::binding` equal to `attribDesc.binding`.
- Let `stride` be the member of `VkPipelineVertexInputStateCreateInfo::pVertexBindingDescriptions->stride` unless there is dynamic state causing the value to be ignored. In this case the value is set from the last value from one of the following
 - `vkCmdSetVertexInputEXT::pVertexBindingDescriptions->stride`
 - `vkCmdBindVertexBuffers2EXT::pStrides`, if not `NULL`

```
bufferBindingAddress = buffer[binding].baseAddress + offset[binding];

if (bindingDesc.inputRate == VK_VERTEX_INPUT_RATE_VERTEX)
    effectiveVertexOffset = vertexIndex * stride;
else
    if (divisor == 0)
        effectiveVertexOffset = firstInstance * stride;
    else
        effectiveVertexOffset = (firstInstance + ((instanceIndex - firstInstance) /
divisor)) * stride;

attribAddress = bufferBindingAddress + effectiveVertexOffset + attribDesc.offset;
```

22.4.1. Vertex Input Extraction

For each attribute, raw data is extracted starting at `attribAddress` and is converted from the `VkVertexInputAttributeDescription`'s `format` in the same manner as described for [image reads](#) as if a texel were read from that address. The numeric type of the attribute's `format` **must** match the numeric type of the input variable in the shader. The input variable in the shader **must** be declared as a 64-bit data type if and only if `format` is a 64-bit data type. If the sum of `attribAddress` and the data extracted is outside of the bound vertex buffer, behavior is as described by [Shader Out-of-Bounds Memory Access](#).

If `format` is a packed format, `attribAddress` **must** be a multiple of the size in bytes of the size of the format as described in [Packed Formats](#). Otherwise, `attribAddress` **must** be a multiple of the size in bytes of the component type indicated by `format` (see [Formats](#)).

The number of components in the vertex shader input variable need not exactly match the number of components in the format. If the vertex shader has fewer components, the extra components are discarded. If the [numeric format](#) of `format` uses sRGB encoding, the implementation **should** convert values from nonlinear to linear as described in the “sRGB EOTF” section of the [Khronos Data Format Specification](#). Implementations which do not convert nonlinear to linear for sRGB formats **should** not expose `VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT` for such formats.

Chapter 23. Tessellation

Tessellation involves three pipeline stages. First, a [tessellation control shader](#) transforms control points of a patch and **can** produce per-patch data. Second, a fixed-function tessellator generates multiple primitives corresponding to a tessellation of the patch in (u,v) or (u,v,w) parameter space. Third, a [tessellation evaluation shader](#) transforms the vertices of the tessellated patch, for example to compute their positions and attributes as part of the tessellated surface. The tessellator is enabled when the pipeline contains both a tessellation control shader and a tessellation evaluation shader.

23.1. Tessellator

If a pipeline includes both tessellation shaders (control and evaluation), the tessellator consumes each input patch (after vertex shading) and produces a new set of independent primitives (points, lines, or triangles). These primitives are logically produced by subdividing a geometric primitive (rectangle or triangle) according to the per-patch outer and inner tessellation levels written by the tessellation control shader. These levels are specified using the [built-in variables](#) `TessLevelOuter` and `TessLevelInner`, respectively. This subdivision is performed in an implementation-dependent manner. If no tessellation shaders are present in the pipeline, the tessellator is disabled and incoming primitives are passed through without modification.

The type of subdivision performed by the tessellator is specified by an `OpExecutionMode` instruction using one of the `Triangles`, `Quads`, or `Isolines` execution modes. This instruction **may** be specified in either the tessellation evaluation or tessellation control shader. Other tessellation-related execution modes **can** also be specified in either the tessellation control or tessellation evaluation shaders.

Any tessellation-related modes specified in both the tessellation control and tessellation evaluation shaders **must** be the same.

Tessellation execution modes include:

- `Triangles`, `Quads`, and `Isolines`. These control the type of subdivision and topology of the output primitives. One mode **must** be set in at least one of the tessellation shader stages.
- `VertexOrderCw` and `VertexOrderCcw`. These control the orientation of triangles generated by the tessellator. One mode **must** be set in at least one of the tessellation shader stages.
- `PointMode`. Controls generation of points rather than triangles or lines. This functionality defaults to disabled, and is enabled if either shader stage includes the execution mode.
- `SpacingEqual`, `SpacingFractionalEven`, and `SpacingFractionalOdd`. Controls the spacing of segments on the edges of tessellated primitives. One mode **must** be set in at least one of the tessellation shader stages.
- `OutputVertices`. Controls the size of the output patch of the tessellation control shader. One value **must** be set in at least one of the tessellation shader stages.

For triangles, the tessellator subdivides a triangle primitive into smaller triangles. For quads, the tessellator subdivides a rectangle primitive into smaller triangles. For isolines, the tessellator subdivides a rectangle primitive into a collection of line segments arranged in strips stretching

across the rectangle in the u dimension (i.e. the coordinates in **TessCoord** are of the form (0,x) through (1,x) for all tessellation evaluation shader invocations that share a line).

Each vertex produced by the tessellator has an associated (u,v,w) or (u,v) position in a normalized parameter space, with parameter values in the range [0,1], as illustrated in figures [Domain parameterization for tessellation primitive modes \(upper-left origin\)](#) and [Domain parameterization for tessellation primitive modes \(lower-left origin\)](#). The domain space **can** have either an upper-left or lower-left origin, selected by the **domainOrigin** member of [VkPipelineTessellationDomainOriginStateCreateInfo](#).

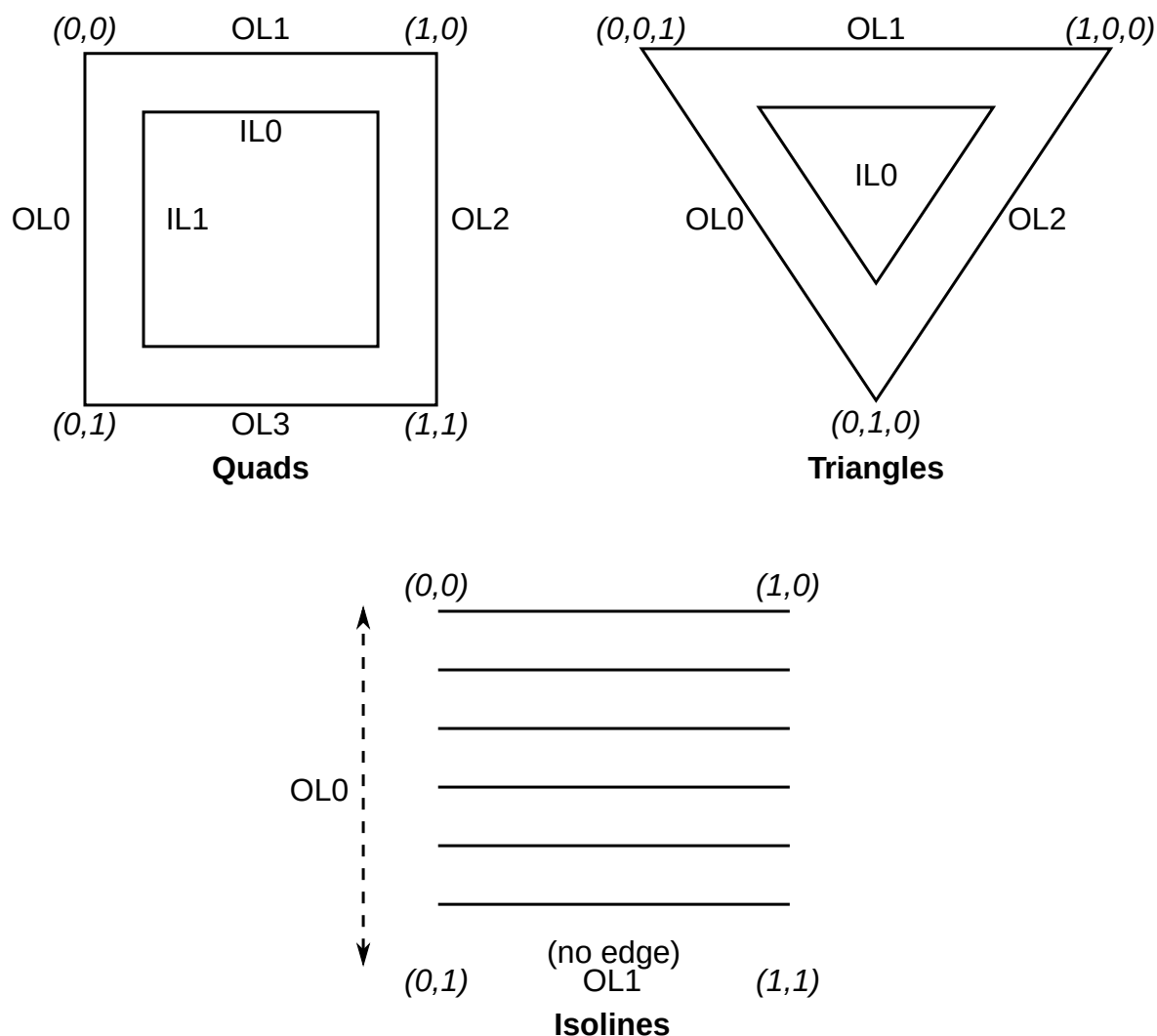


Figure 11. Domain parameterization for tessellation primitive modes (upper-left origin)

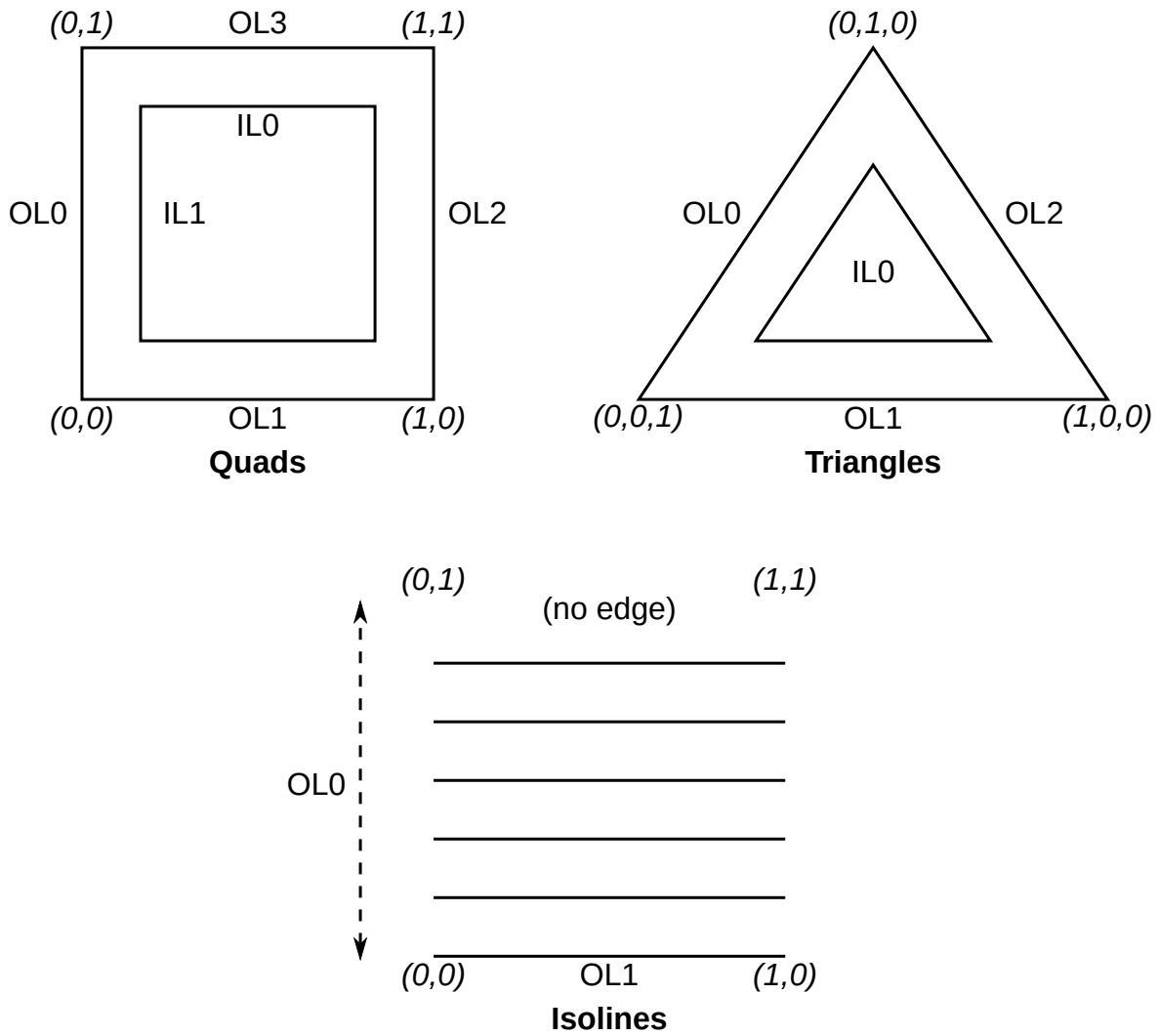


Figure 12. Domain parameterization for tessellation primitive modes (lower-left origin)

Caption

In the domain parameterization diagrams, the coordinates illustrate the value of **TessCoord** at the corners of the domain. The labels on the edges indicate the inner (IL0 and IL1) and outer (OL0 through OL3) tessellation level values used to control the number of subdivisions along each edge of the domain.

For triangles, the vertex's position is a barycentric coordinate (u,v,w) , where $u + v + w = 1.0$, and indicates the relative influence of the three vertices of the triangle on the position of the vertex. For quads and isolines, the position is a (u,v) coordinate indicating the relative horizontal and vertical position of the vertex relative to the subdivided rectangle. The subdivision process is explained in more detail in subsequent sections.

23.2. Tessellator Patch Discard

A patch is discarded by the tessellator if any relevant outer tessellation level is less than or equal to zero.

Patches will also be discarded if any relevant outer tessellation level corresponds to a floating-point

NaN (not a number) in implementations supporting NaN.

No new primitives are generated and the tessellation evaluation shader is not executed for patches that are discarded. For **Quads**, all four outer levels are relevant. For **Triangles** and **Isolines**, only the first three or two outer levels, respectively, are relevant. Negative inner levels will not cause a patch to be discarded; they will be clamped as described below.

23.3. Tessellator Spacing

Each of the tessellation levels is used to determine the number and spacing of segments used to subdivide a corresponding edge. The method used to derive the number and spacing of segments is specified by an **OpExecutionMode** in the tessellation control or tessellation evaluation shader using one of the identifiers **SpacingEqual**, **SpacingFractionalEven**, or **SpacingFractionalOdd**.

If **SpacingEqual** is used, the floating-point tessellation level is first clamped to $[1, \text{maxLevel}]$, where **maxLevel** is the implementation-dependent maximum tessellation level (**VkPhysicalDeviceLimits::maxTessellationGenerationLevel**). The result is rounded up to the nearest integer n , and the corresponding edge is divided into n segments of equal length in (u,v) space.

If **SpacingFractionalEven** is used, the tessellation level is first clamped to $[2, \text{maxLevel}]$ and then rounded up to the nearest even integer n . If **SpacingFractionalOdd** is used, the tessellation level is clamped to $[1, \text{maxLevel} - 1]$ and then rounded up to the nearest odd integer n . If n is one, the edge will not be subdivided. Otherwise, the corresponding edge will be divided into $n - 2$ segments of equal length, and two additional segments of equal length that are typically shorter than the other segments. The length of the two additional segments relative to the others will decrease monotonically with $n - f$, where f is the clamped floating-point tessellation level. When $n - f$ is zero, the additional segments will have equal length to the other segments. As $n - f$ approaches 2.0, the relative length of the additional segments approaches zero. The two additional segments **must** be placed symmetrically on opposite sides of the subdivided edge. The relative location of these two segments is implementation-dependent, but **must** be identical for any pair of subdivided edges with identical values of f .

When tessellating triangles or quads using **point mode** with fractional odd spacing, the tessellator **may** produce *interior vertices* that are positioned on the edge of the patch if an inner tessellation level is less than or equal to one. Such vertices are considered distinct from vertices produced by subdividing the outer edge of the patch, even if there are pairs of vertices with identical coordinates.

23.4. Tessellation Primitive Ordering

Few guarantees are provided for the relative ordering of primitives produced by tessellation, as they pertain to **primitive order**.

- The output primitives generated from each input primitive are passed to subsequent pipeline stages in an implementation-dependent order.
- All output primitives generated from a given input primitive are passed to subsequent pipeline stages before any output primitives generated from subsequent input primitives.

23.5. Tessellator Vertex Winding Order

When the tessellator produces triangles (in the **Triangles** or **Quads** modes), the orientation of all triangles is specified with an **OpExecutionMode** of **VertexOrderCw** or **VertexOrderCcw** in the tessellation control or tessellation evaluation shaders. If the order is **VertexOrderCw**, the vertices of all generated triangles will have clockwise ordering in (u,v) or (u,v,w) space. If the order is **VertexOrderCcw**, the vertices will have counter-clockwise ordering in that space.

If the tessellation domain has an upper-left origin, the vertices of a triangle have counter-clockwise ordering if

$$a = u_0 v_1 - u_1 v_0 + u_1 v_2 - u_2 v_1 + u_2 v_0 - u_0 v_2$$

is negative, and clockwise ordering if a is positive. u_i and v_i are the u and v coordinates in normalized parameter space of the i th vertex of the triangle. If the tessellation domain has a lower-left origin, the vertices of a triangle have counter-clockwise ordering if a is positive, and clockwise ordering if a is negative.

NOTE

The value a is proportional (with a positive factor) to the signed area of the triangle.

In **Triangles** mode, even though the vertex coordinates have a w value, it does not participate directly in the computation of a , being an affine combination of u and v .

23.6. Triangle Tessellation

If the tessellation primitive mode is **Triangles**, an equilateral triangle is subdivided into a collection of triangles covering the area of the original triangle. First, the original triangle is subdivided into a collection of concentric equilateral triangles. The edges of each of these triangles are subdivided, and the area between each triangle pair is filled by triangles produced by joining the vertices on the subdivided edges. The number of concentric triangles and the number of subdivisions along each triangle except the outermost is derived from the first inner tessellation level. The edges of the outermost triangle are subdivided independently, using the first, second, and third outer tessellation levels to control the number of subdivisions of the $u = 0$ (left), $v = 0$ (bottom), and $w = 0$ (right) edges, respectively. The second inner tessellation level and the fourth outer tessellation level have no effect in this mode.

If the first inner tessellation level and all three outer tessellation levels are exactly one after clamping and rounding, only a single triangle with (u,v,w) coordinates of (0,0,1), (1,0,0), and (0,1,0) is generated. If the inner tessellation level is one and any of the outer tessellation levels is greater than one, the inner tessellation level is treated as though it were originally specified as $1 + \epsilon$ and will result in a two- or three-segment subdivision depending on the tessellation spacing. When used with fractional odd spacing, the three-segment subdivision **may** produce *inner vertices* positioned on the edge of the triangle.

If any tessellation level is greater than one, tessellation begins by producing a set of concentric inner triangles and subdividing their edges. First, the three outer edges are temporarily subdivided using the clamped and rounded first inner tessellation level and the specified tessellation spacing,

generating n segments. For the outermost inner triangle, the inner triangle is degenerate — a single point at the center of the triangle — if n is two. Otherwise, for each corner of the outer triangle, an inner triangle corner is produced at the intersection of two lines extended perpendicular to the corner's two adjacent edges running through the vertex of the subdivided outer edge nearest that corner. If n is three, the edges of the inner triangle are not subdivided and it is the final triangle in the set of concentric triangles. Otherwise, each edge of the inner triangle is divided into $n - 2$ segments, with the $n - 1$ vertices of this subdivision produced by intersecting the inner edge with lines perpendicular to the edge running through the $n - 1$ innermost vertices of the subdivision of the outer edge. Once the outermost inner triangle is subdivided, the previous subdivision process repeats itself, using the generated triangle as an outer triangle. This subdivision process is illustrated in [Inner Triangle Tessellation](#).

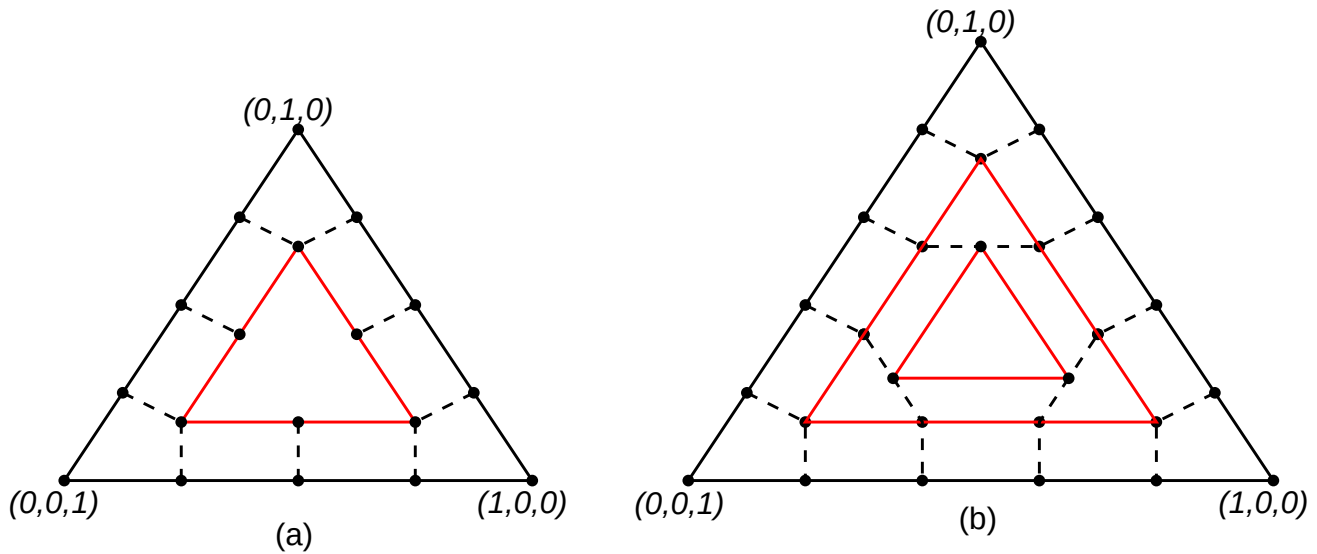


Figure 13. Inner Triangle Tessellation

Caption

In the [Inner Triangle Tessellation](#) diagram, inner tessellation levels of (a) four and (b) five are shown (not to scale). Solid black circles depict vertices along the edges of the concentric triangles. The edges of inner triangles are subdivided by intersecting the edge with segments perpendicular to the edge passing through each inner vertex of the subdivided outer edge. Dotted lines depict edges connecting corresponding vertices on the inner and outer triangle edges.

Once all the concentric triangles are produced and their edges are subdivided, the area between each pair of adjacent inner triangles is filled completely with a set of non-overlapping triangles. In this subdivision, two of the three vertices of each triangle are taken from adjacent vertices on a subdivided edge of one triangle; the third is one of the vertices on the corresponding edge of the other triangle. If the innermost triangle is degenerate (i.e., a point), the triangle containing it is subdivided into six triangles by connecting each of the six vertices on that triangle with the center point. If the innermost triangle is not degenerate, that triangle is added to the set of generated triangles as-is.

After the area corresponding to any inner triangles is filled, the tessellator generates triangles to cover the area between the outermost triangle and the outermost inner triangle. To do this, the

temporary subdivision of the outer triangle edge above is discarded. Instead, the $u = 0$, $v = 0$, and $w = 0$ edges are subdivided according to the first, second, and third outer tessellation levels, respectively, and the tessellation spacing. The original subdivision of the first inner triangle is retained. The area between the outer and first inner triangles is completely filled by non-overlapping triangles as described above. If the first (and only) inner triangle is degenerate, a set of triangles is produced by connecting each vertex on the outer triangle edges with the center point.

After all triangles are generated, each vertex in the subdivided triangle is assigned a barycentric (u,v,w) coordinate based on its location relative to the three vertices of the outer triangle.

The algorithm used to subdivide the triangular domain in (u,v,w) space into individual triangles is implementation-dependent. However, the set of triangles produced will completely cover the domain, and no portion of the domain will be covered by multiple triangles.

Output triangles are generated with a topology similar to [triangle lists](#), except that the order in which each triangle is generated, and the order in which the vertices are generated for each triangle, are implementation-dependent. However, the order of vertices in each triangle is consistent across the domain as described in [Tessellator Vertex Winding Order](#).

23.7. Quad Tessellation

If the tessellation primitive mode is **Quads**, a rectangle is subdivided into a collection of triangles covering the area of the original rectangle. First, the original rectangle is subdivided into a regular mesh of rectangles, where the number of rectangles along the $u = 0$ and $u = 1$ (vertical) and $v = 0$ and $v = 1$ (horizontal) edges are derived from the first and second inner tessellation levels, respectively. All rectangles, except those adjacent to one of the outer rectangle edges, are decomposed into triangle pairs. The outermost rectangle edges are subdivided independently, using the first, second, third, and fourth outer tessellation levels to control the number of subdivisions of the $u = 0$ (left), $v = 0$ (bottom), $u = 1$ (right), and $v = 1$ (top) edges, respectively. The area between the inner rectangles of the mesh and the outer rectangle edges are filled by triangles produced by joining the vertices on the subdivided outer edges to the vertices on the edge of the inner rectangle mesh.

If both clamped inner tessellation levels and all four clamped outer tessellation levels are exactly one, only a single triangle pair covering the outer rectangle is generated. Otherwise, if either clamped inner tessellation level is one, that tessellation level is treated as though it was originally specified as $1 + \epsilon$ and will result in a two- or three-segment subdivision depending on the tessellation spacing. When used with fractional odd spacing, the three-segment subdivision **may** produce *inner vertices* positioned on the edge of the rectangle.

If any tessellation level is greater than one, tessellation begins by subdividing the $u = 0$ and $u = 1$ edges of the outer rectangle into m segments using the clamped and rounded first inner tessellation level and the tessellation spacing. The $v = 0$ and $v = 1$ edges are subdivided into n segments using the second inner tessellation level. Each vertex on the $u = 0$ and $v = 0$ edges are joined with the corresponding vertex on the $u = 1$ and $v = 1$ edges to produce a set of vertical and horizontal lines that divide the rectangle into a grid of smaller rectangles. The primitive generator emits a pair of non-overlapping triangles covering each such rectangle not adjacent to an edge of the outer rectangle. The boundary of the region covered by these triangles forms an inner rectangle, the

edges of which are subdivided by the grid vertices that lie on the edge. If either m or n is two, the inner rectangle is degenerate, and one or both of the rectangle's *edges* consist of a single point. This subdivision is illustrated in Figure [Inner Quad Tessellation](#).

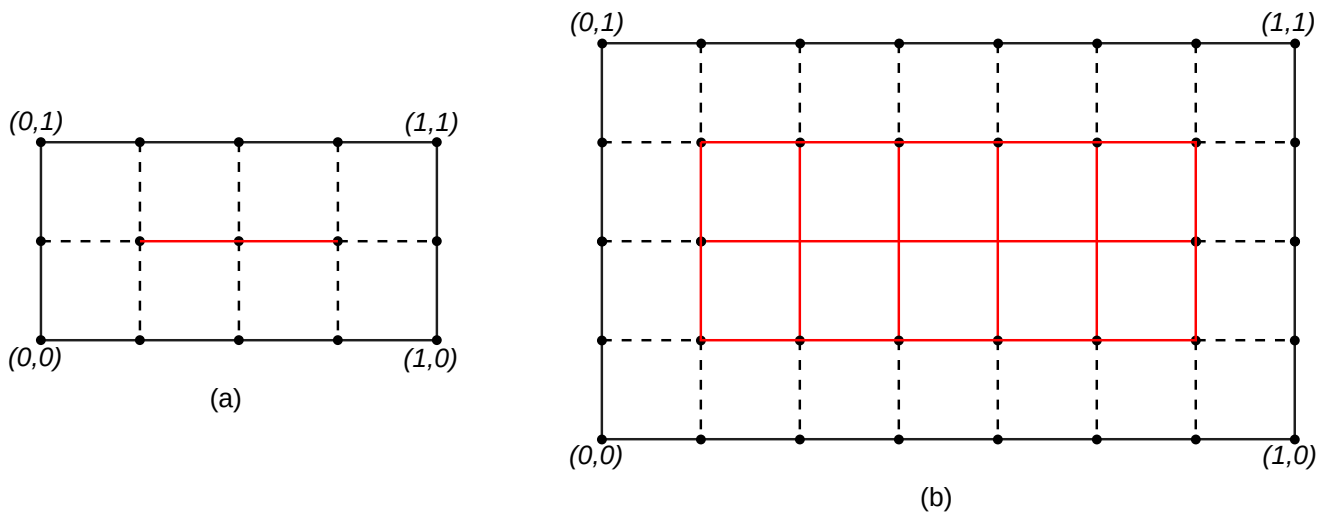


Figure 14. Inner Quad Tessellation

Caption

In the [Inner Quad Tessellation](#) diagram, inner quad tessellation levels of (a) (4,2) and (b) (7,4) are shown. The regions highlighted in red in figure (b) depict the 10 inner rectangles, each of which will be subdivided into two triangles. Solid black circles depict vertices on the boundary of the outer and inner rectangles, where the inner rectangle of figure (a) is degenerate (a single line segment). Dotted lines depict the horizontal and vertical edges connecting corresponding vertices on the inner and outer rectangle edges.

After the area corresponding to the inner rectangle is filled, the tessellator **must** produce triangles to cover the area between the inner and outer rectangles. To do this, the subdivision of the outer rectangle edge above is discarded. Instead, the $u = 0$, $v = 0$, $u = 1$, and $v = 1$ edges are subdivided according to the first, second, third, and fourth outer tessellation levels, respectively, and the tessellation spacing. The original subdivision of the inner rectangle is retained. The area between the outer and inner rectangles is completely filled by non-overlapping triangles. Two of the three vertices of each triangle are adjacent vertices on a subdivided edge of one rectangle; the third is one of the vertices on the corresponding edge of the other rectangle. If either edge of the innermost rectangle is degenerate, the area near the corresponding outer edges is filled by connecting each vertex on the outer edge with the single vertex making up the *inner edge*.

The algorithm used to subdivide the rectangular domain in (u,v) space into individual triangles is implementation-dependent. However, the set of triangles produced will completely cover the domain, and no portion of the domain will be covered by multiple triangles.

Output triangles are generated with a topology similar to [triangle lists](#), except that the order in which each triangle is generated, and the order in which the vertices are generated for each triangle, are implementation-dependent. However, the order of vertices in each triangle is consistent across the domain as described in [Tessellator Vertex Winding Order](#).

23.8. Isoline Tessellation

If the tessellation primitive mode is **Isolines**, a set of independent horizontal line segments is drawn. The segments are arranged into connected strips called *isolines*, where the vertices of each isoline have a constant *v* coordinate and *u* coordinates covering the full range [0,1]. The number of isolines generated is derived from the first outer tessellation level; the number of segments in each isoline is derived from the second outer tessellation level. Both inner tessellation levels and the third and fourth outer tessellation levels have no effect in this mode.

As with quad tessellation above, isoline tessellation begins with a rectangle. The *u* = 0 and *u* = 1 edges of the rectangle are subdivided according to the first outer tessellation level. For the purposes of this subdivision, the tessellation spacing mode is ignored and treated as `equal_spacing`. An isoline is drawn connecting each vertex on the *u* = 0 rectangle edge to the corresponding vertex on the *u* = 1 rectangle edge, except that no line is drawn between (0,1) and (1,1). If the number of isolines on the subdivided *u* = 0 and *u* = 1 edges is *n*, this process will result in *n* equally spaced lines with constant *v* coordinates of $0, \frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}$.

Each of the *n* isolines is then subdivided according to the second outer tessellation level and the tessellation spacing, resulting in *m* line segments. Each segment of each line is emitted by the tessellator. These line segments are generated with a topology similar to [line lists](#), except that the order in which each line is generated, and the order in which the vertices are generated for each line segment, are implementation-dependent.

23.9. Tessellation Point Mode

For all primitive modes, the tessellator is capable of generating points instead of lines or triangles. If the tessellation control or tessellation evaluation shader specifies the **OpExecutionMode PointMode**, the primitive generator will generate one point for each distinct vertex produced by tessellation, rather than emitting triangles or lines. Otherwise, the tessellator will produce a collection of line segments or triangles according to the primitive mode. These points are generated with a topology similar to [point lists](#), except the order in which the points are generated for each input primitive is undefined.

23.10. Tessellation Pipeline State

The `pTessellationState` member of [VkGraphicsPipelineCreateInfo](#) is a pointer to a [VkPipelineTessellationStateCreateInfo](#) structure.

The [VkPipelineTessellationStateCreateInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineTessellationStateCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkPipelineTessellationStateCreateFlags flags;
    uint32_t                  patchControlPoints;
} VkPipelineTessellationStateCreateInfo;
```


- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `patchControlPoints` is the number of control points per patch.

Valid Usage

- VUID-VkPipelineTessellationStateCreateInfo-patchControlPoints-01214
`patchControlPoints` **must** be greater than zero and less than or equal to `VkPhysicalDeviceLimits::maxTessellationPatchSize`

Valid Usage (Implicit)

- VUID-VkPipelineTessellationStateCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_STATE_CREATE_INFO`
- VUID-VkPipelineTessellationStateCreateInfo-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkPipelineTessellationDomainOriginStateCreateInfo`
- VUID-VkPipelineTessellationStateCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkPipelineTessellationStateCreateInfo-flags-zero bitmask
`flags` **must** be 0

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineTessellationStateCreateFlags;
```

`VkPipelineTessellationStateCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

The `VkPipelineTessellationDomainOriginStateCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPipelineTessellationDomainOriginStateCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkTessellationDomainOrigin domainOrigin;
} VkPipelineTessellationDomainOriginStateCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `domainOrigin` is a `VkTessellationDomainOrigin` value controlling the origin of the tessellation

domain space.

If the `VkPipelineTessellationDomainOriginStateCreateInfo` structure is included in the `pNext` chain of `VkPipelineTessellationStateCreateInfo`, it controls the origin of the tessellation domain. If this structure is not present, it is as if `domainOrigin` was `VK_TESSELLATION_DOMAIN_ORIGIN_UPPER_LEFT`.

Valid Usage (Implicit)

- VUID-VkPipelineTessellationDomainOriginStateCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_DOMAIN_ORIGIN_STATE_CREATE_INFO`
- VUID-VkPipelineTessellationDomainOriginStateCreateInfo-domainOrigin-parameter
`domainOrigin` **must** be a valid `VkTessellationDomainOrigin` value

The possible tessellation domain origins are specified by the `VkTessellationDomainOrigin` enumeration:

```
// Provided by VK_VERSION_1_1
typedef enum VkTessellationDomainOrigin {
    VK_TESSELLATION_DOMAIN_ORIGIN_UPPER_LEFT = 0,
    VK_TESSELLATION_DOMAIN_ORIGIN_LOWER_LEFT = 1,
} VkTessellationDomainOrigin;
```

- `VK_TESSELLATION_DOMAIN_ORIGIN_UPPER_LEFT` specifies that the origin of the domain space is in the upper left corner, as shown in figure [Domain parameterization for tessellation primitive modes \(upper-left origin\)](#).
- `VK_TESSELLATION_DOMAIN_ORIGIN_LOWER_LEFT` specifies that the origin of the domain space is in the lower left corner, as shown in figure [Domain parameterization for tessellation primitive modes \(lower-left origin\)](#).

This enum affects how the `VertexOrderCw` and `VertexOrderCcw` tessellation execution modes are interpreted, since the winding is defined relative to the orientation of the domain.

Chapter 24. Geometry Shading

The geometry shader operates on a group of vertices and their associated data assembled from a single input primitive, and emits zero or more output primitives and the group of vertices and their associated data required for each output primitive. Geometry shading is enabled when a geometry shader is included in the pipeline.

24.1. Geometry Shader Input Primitives

Each geometry shader invocation has access to all vertices in the primitive (and their associated data), which are presented to the shader as an array of inputs.

The input primitive type expected by the geometry shader is specified with an `OpExecutionMode` instruction in the geometry shader, and **must** match the incoming primitive type specified by either the pipeline's `primitive topology` if tessellation is inactive, or the `tessellation mode` if tessellation is active, as follows:

- An input primitive type of `InputPoints` **must** only be used with a pipeline topology of `VK_PRIMITIVE_TOPOLOGY_POINT_LIST`, or with a tessellation shader specifying `PointMode`. The input arrays always contain one element, as described by the `point list topology` or `tessellation in point mode`.
- An input primitive type of `InputLines` **must** only be used with a pipeline topology of `VK_PRIMITIVE_TOPOLOGY_LINE_LIST` or `VK_PRIMITIVE_TOPOLOGY_LINE_STRIP`, or with a tessellation shader specifying `IsoLines` that does not specify `PointMode`. The input arrays always contain two elements, as described by the `line list topology` or `line strip topology`, or by `isoline tessellation`.
- An input primitive type of `InputLinesAdjacency` **must** only be used when tessellation is inactive, with a pipeline topology of `VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY` or `VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY`. The input arrays always contain four elements, as described by the `line list with adjacency topology` or `line strip with adjacency topology`.
- An input primitive type of `Triangles` **must** only be used with a pipeline topology of `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST`, `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP`, or `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN`; or with a tessellation shader specifying `Quads` or `Triangles` that does not specify `PointMode`. The input arrays always contain three elements, as described by the `triangle list topology`, `triangle strip topology`, or `triangle fan topology`, or by `triangle` or `quad tessellation`. Vertices **may** be in a different absolute order than specified by the topology, but **must** adhere to the specified winding order.
- An input primitive type of `InputTrianglesAdjacency` **must** only be used when tessellation is inactive, with a pipeline topology of `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY` or `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY`. The input arrays always contain six elements, as described by the `triangle list with adjacency topology` or `triangle strip with adjacency topology`. Vertices **may** be in a different absolute order than specified by the topology, but **must** adhere to the specified winding order, and the vertices making up the main primitive **must** still occur at the first, third, and fifth index.

24.2. Geometry Shader Output Primitives

A geometry shader generates primitives in one of three output modes: points, line strips, or triangle strips. The primitive mode is specified in the shader using an `OpExecutionMode` instruction with the `OutputPoints`, `OutputLineStrip` or `OutputTriangleStrip` modes, respectively. Each geometry shader **must** include exactly one output primitive mode.

The vertices output by the geometry shader are assembled into points, lines, or triangles based on the output primitive type and the resulting primitives are then further processed as described in [Rasterization](#). If the number of vertices emitted by the geometry shader is not sufficient to produce a single primitive, vertices corresponding to incomplete primitives are not processed by subsequent pipeline stages. The number of vertices output by the geometry shader is limited to a maximum count specified in the shader.

The maximum output vertex count is specified in the shader using an `OpExecutionMode` instruction with the mode set to `OutputVertices` and the maximum number of vertices that will be produced by the geometry shader specified as a literal. Each geometry shader **must** specify a maximum output vertex count.

24.3. Multiple Invocations of Geometry Shaders

Geometry shaders **can** be invoked more than one time for each input primitive. This is known as *geometry shader instancing* and is requested by including an `OpExecutionMode` instruction with `mode` specified as `Invocations` and the number of invocations specified as an integer literal.

In this mode, the geometry shader will execute at least *n* times for each input primitive, where *n* is the number of invocations specified in the `OpExecutionMode` instruction. The instance number is available to each invocation as a built-in input using `InvocationId`.

24.4. Geometry Shader Primitive Ordering

Limited guarantees are provided for the relative ordering of primitives produced by a geometry shader, as they pertain to [primitive order](#).

- For instanced geometry shaders, the output primitives generated from each input primitive are passed to subsequent pipeline stages using the invocation number to order the primitives, from least to greatest.
- All output primitives generated from a given input primitive are passed to subsequent pipeline stages before any output primitives generated from subsequent input primitives.

Chapter 25. Fixed-Function Vertex Post-Processing

After [pre-rasterization shader stages](#), the following fixed-function operations are applied to vertices of the resulting primitives:

- Flat shading (see [Flat Shading](#)).
- Primitive clipping, including application-defined half-spaces (see [Primitive Clipping](#)).
- Shader output attribute clipping (see [Clipping Shader Outputs](#)).
- Perspective division on clip coordinates (see [Coordinate Transformations](#)).
- Viewport mapping, including depth range scaling (see [Controlling the Viewport](#)).
- Front face determination for polygon primitives (see [Basic Polygon Rasterization](#)).

Next, rasterization is performed on primitives as described in chapter [Rasterization](#).

25.1. Flat Shading

Flat shading a vertex output attribute means to assign all vertices of the primitive the same value for that output. The output values assigned are those of the *provoking vertex* of the primitive. Flat shading is applied to those vertex attributes that [match](#) fragment input attributes which are decorated as **Flat**.

If neither [geometry](#) nor [tessellation shading](#) is active, the provoking vertex is determined by the [primitive topology](#) defined by `VkPipelineInputAssemblyStateCreateInfo:topology` used to execute the [drawing command](#).

If [geometry shading](#) is active, the provoking vertex is determined by the [primitive topology](#) defined by the `OutputPoints`, `OutputLineStrip`, or `OutputTriangleStrip` execution mode.

If [tessellation shading](#) is active but [geometry shading](#) is not, the provoking vertex **may** be any of the vertices in each primitive.

25.2. Primitive Clipping

Primitives are culled against the *cull volume* and then clipped to the *clip volume*. In clip coordinates, the *view volume* is defined by:

$$\begin{aligned} -w_c &\leq x_c \leq w_c \\ -w_c &\leq y_c \leq w_c \\ z_m &\leq z_c \leq w_c \end{aligned}$$

where z_m is equal to zero.

This view volume **can** be further restricted by as many as `VkPhysicalDeviceLimits::maxClipDistances` application-defined half-spaces.

The cull volume is the intersection of up to `VkPhysicalDeviceLimits::maxCullDistances` application-defined half-spaces (if no application-defined cull half-spaces are enabled, culling against the cull volume is skipped).

A shader **must** write a single cull distance for each enabled cull half-space to elements of the `CullDistance` array. If the cull distance for any enabled cull half-space is negative for all of the vertices of the primitive under consideration, the primitive is discarded. Otherwise the primitive is clipped against the clip volume as defined below.

The clip volume is the intersection of up to `VkPhysicalDeviceLimits::maxClipDistances` application-defined half-spaces with the view volume (if no application-defined clip half-spaces are enabled, the clip volume is the view volume).

A shader **must** write a single clip distance for each enabled clip half-space to elements of the `ClipDistance` array. Clip half-space i is then given by the set of points satisfying the inequality

$$c_i(\mathbf{P}) \geq 0$$

where $c_i(\mathbf{P})$ is the clip distance i at point $\mathbf{P}$. For point primitives, $c_i(\mathbf{P})$ is simply the clip distance for the vertex in question. For line and triangle primitives, per-vertex clip distances are interpolated using a weighted mean, with weights derived according to the algorithms described in sections [Basic Line Segment Rasterization](#) and [Basic Polygon Rasterization](#), using the perspective interpolation equations.

The number of application-defined clip and cull half-spaces that are enabled is determined by the explicit size of the built-in arrays `ClipDistance` and `CullDistance`, respectively, declared as an output in the interface of the entry point of the final shader stage before clipping.

If `VkPipelineRasterizationDepthClipStateCreateInfoEXT` is present in the graphics pipeline state then depth clipping is disabled if `VkPipelineRasterizationDepthClipStateCreateInfoEXT::depthClipEnable` is `VK_FALSE`. Otherwise, if `VkPipelineRasterizationDepthClipStateCreateInfoEXT` is not present, depth clipping is disabled when `VkPipelineRasterizationStateCreateInfo::depthClampEnable` is `VK_TRUE`.

When depth clipping is disabled, the plane equation

$$Z_m \leq Z_c \leq W_c$$

(see the clip volume definition above) is ignored by view volume clipping (effectively, there is no near or far plane clipping).

If the primitive under consideration is a point or line segment, then clipping passes it unchanged if its vertices lie entirely within the clip volume.

Possible values of `VkPhysicalDevicePointClippingProperties::pointClippingBehavior`, specifying clipping behavior of a point primitive whose vertex lies outside the clip volume, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkPointClippingBehavior {
    VK_POINT_CLIPPING_BEHAVIOR_ALL_CLIP_PLANES = 0,
    VK_POINT_CLIPPING_BEHAVIOR_USER_CLIP_PLANES_ONLY = 1,
} VkPointClippingBehavior;
```

- **VK_POINT_CLIPPING_BEHAVIOR_ALL_CLIP_PLANES** specifies that the primitive is discarded if the vertex lies outside any clip plane, including the planes bounding the view volume.
- **VK_POINT_CLIPPING_BEHAVIOR_USER_CLIP_PLANES_ONLY** specifies that the primitive is discarded only if the vertex lies outside any user clip plane.

If either of a line segment's vertices lie outside of the clip volume, the line segment **may** be clipped, with new vertex coordinates computed for each vertex that lies outside the clip volume. A clipped line segment endpoint lies on both the original line segment and the boundary of the clip volume.

This clipping produces a value, $0 \leq t \leq 1$, for each clipped vertex. If the coordinates of a clipped vertex are **P** and the unclipped line segment's vertex coordinates are **P**₁ and **P**₂, then t satisfies the following equation

$$\mathbf{P} = t \mathbf{P}_1 + (1-t) \mathbf{P}_2.$$

t is used to clip vertex output attributes as described in [Clipping Shader Outputs](#).

If the primitive is a polygon, it passes unchanged if every one of its edges lies entirely inside the clip volume, and is either clipped or discarded otherwise. If the edges of the polygon intersect the boundary of the clip volume, the intersecting edges are reconnected by new edges that lie along the boundary of the clip volume - in some cases requiring the introduction of new vertices into a polygon.

If a polygon intersects an edge of the clip volume's boundary, the clipped polygon **must** include a point on this boundary edge.

Primitives rendered with application-defined half-spaces **must** satisfy a complementarity criterion. Suppose a series of primitives is drawn where each vertex i has a single specified clip distance d_i (or a number of similarly specified clip distances, if multiple half-spaces are enabled). Next, suppose that the same series of primitives are drawn again with each such clip distance replaced by $-d_i$ (and the graphics pipeline is otherwise the same). In this case, primitives **must** not be missing any pixels, and pixels **must** not be drawn twice in regions where those primitives are cut by the clip planes.

25.3. Clipping Shader Outputs

Next, vertex output attributes are clipped. The output values associated with a vertex that lies within the clip volume are unaffected by clipping. If a primitive is clipped, however, the output values assigned to vertices produced by clipping are clipped.

Let the output values assigned to the two vertices **P**₁ and **P**₂ of an unclipped edge be **c**₁ and **c**₂. The value of t (see [Primitive Clipping](#)) for a clipped point **P** is used to obtain the output value associated

with $\mathbf{P}$ as

$$\mathbf{c} = t \mathbf{c}_1 + (1-t) \mathbf{c}_2.$$

(Multiplying an output value by a scalar means multiplying each of x , y , z , and w by the scalar.)

Since this computation is performed in clip space before division by w_c , clipped output values are perspective-correct.

Polygon clipping creates a clipped vertex along an edge of the clip volume's boundary. This situation is handled by noting that polygon clipping proceeds by clipping against one half-space at a time. Output value clipping is done in the same way, so that clipped points always occur at the intersection of polygon edges (possibly already clipped) with the clip volume's boundary.

For vertex output attributes whose matching fragment input attributes are decorated with **NoPerspective**, the value of t used to obtain the output value associated with $\mathbf{P}$ will be adjusted to produce results that vary linearly in framebuffer space.

Output attributes of integer or unsigned integer type **must** always be flat shaded. Flat shaded attributes are constant over the primitive being rasterized (see [Basic Line Segment Rasterization](#) and [Basic Polygon Rasterization](#)), and no interpolation is performed. The output value $\mathbf{c}$ is taken from either $\mathbf{c}_1$ or $\mathbf{c}_2$, since flat shading has already occurred and the two values are identical.

25.4. Coordinate Transformations

Clip coordinates for a vertex result from shader execution, which yields a vertex coordinate **Position**.

Perspective division on clip coordinates yields *normalized device coordinates*, followed by a *viewport* transformation (see [Controlling the Viewport](#)) to convert these coordinates into *framebuffer coordinates*.

If a vertex in clip coordinates has a position given by

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix}$$

then the vertex's normalized device coordinates are

$$\begin{pmatrix} x_d \\ y_d \\ z_d \end{pmatrix} = \begin{pmatrix} \frac{x_c}{w_c} \\ \frac{y_c}{w_c} \\ \frac{z_c}{w_c} \end{pmatrix}$$

25.5. Controlling the Viewport

The viewport transformation is determined by the selected viewport's width and height in pixels, p_x and p_y , respectively, and its center (o_x , o_y) (also in pixels), as well as its depth range min and max determining a depth range scale value p_z and a depth range bias value o_z (defined below). The vertex's framebuffer coordinates (x_f , y_f) and depth z_f are given by

$$x_f = (p_x / 2) x_d + o_x$$

$$y_f = (p_y / 2) y_d + o_y$$

$$z_f = p_z \times z_d + o_z$$

Multiple viewports are available, numbered zero up to `VkPhysicalDeviceLimits::maxViewports` minus one. The number of viewports used by a pipeline is controlled by the `viewportCount` member of the `VkPipelineViewportStateCreateInfo` structure used in pipeline creation.

x_f and y_f have limited precision, where the number of fractional bits retained is specified by `VkPhysicalDeviceLimits::subPixelPrecisionBits`. When rasterizing [line segments](#), the number of fractional bits is specified by `VkPhysicalDeviceLineRasterizationProperties::lineSubPixelPrecisionBits`.

The `VkPipelineViewportStateCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineViewportStateCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkPipelineViewportStateCreateFlags flags;
    uint32_t                 viewportCount;
    const VkViewport*        pViewports;
    uint32_t                 scissorCount;
    const VkRect2D*          pScissors;
} VkPipelineViewportStateCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `viewportCount` is the number of viewports used by the pipeline.
- `pViewports` is a pointer to an array of `VkViewport` structures, defining the viewport transforms. If the viewport state is dynamic, this member is ignored.
- `scissorCount` is the number of `scissors` and **must** match the number of viewports.
- `pScissors` is a pointer to an array of `VkRect2D` structures defining the rectangular bounds of the

scissor for the corresponding viewport. If the scissor state is dynamic, this member is ignored.

Valid Usage

- VUID-VkPipelineViewportStateCreateInfo-viewportCount-01216
If the `multiViewport` feature is not enabled, `viewportCount` **must** not be greater than 1
- VUID-VkPipelineViewportStateCreateInfo-scissorCount-01217
If the `multiViewport` feature is not enabled, `scissorCount` **must** not be greater than 1
- VUID-VkPipelineViewportStateCreateInfo-viewportCount-01218
`viewportCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxViewports`
- VUID-VkPipelineViewportStateCreateInfo-scissorCount-01219
`scissorCount` **must** be less than or equal to `VkPhysicalDeviceLimits::maxViewports`
- VUID-VkPipelineViewportStateCreateInfo-x-02821
The `x` and `y` members of `offset` member of any element of `pScissors` **must** be greater than or equal to 0
- VUID-VkPipelineViewportStateCreateInfo-offset-02822
Evaluation of $(\text{offset.x} + \text{extent.width})$ **must** not cause a signed integer addition overflow for any element of `pScissors`
- VUID-VkPipelineViewportStateCreateInfo-offset-02823
Evaluation of $(\text{offset.y} + \text{extent.height})$ **must** not cause a signed integer addition overflow for any element of `pScissors`
- VUID-VkPipelineViewportStateCreateInfo-scissorCount-04134
If `scissorCount` and `viewportCount` are both not dynamic, then `scissorCount` and `viewportCount` **must** be identical
- VUID-VkPipelineViewportStateCreateInfo-viewportCount-04135
If the graphics pipeline is being created with `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` set then `viewportCount` **must** be 0, otherwise `viewportCount` **must** be greater than 0
- VUID-VkPipelineViewportStateCreateInfo-scissorCount-04136
If the graphics pipeline is being created with `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` set then `scissorCount` **must** be 0, otherwise `scissorCount` **must** be greater than 0

Valid Usage (Implicit)

- VUID-VkPipelineViewportStateCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO`
- VUID-VkPipelineViewportStateCreateInfo-pNext-pNext
`pNext` **must** be NULL
- VUID-VkPipelineViewportStateCreateInfo-flags-zeroBitmask
`flags` **must** be 0

To [dynamically set](#) the viewport count and viewports, call:

```
// Provided by VK_EXT_extended_dynamic_state
void vkCmdSetViewportWithCountEXT(
    VkCommandBuffer                      commandBuffer,
    uint32_t                             viewportCount,
    const VkViewport*                   pViewports);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `viewportCount` specifies the viewport count.
- `pViewports` specifies the viewports to use for drawing.

This command sets the viewport count and viewports state for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the corresponding `VkPipelineViewportStateCreateInfo::viewportCount` and `pViewports` values used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetViewportWithCount-None-08971
At least one of the following **must** be true:
 - the `extendedDynamicState` feature is enabled
- VUID-vkCmdSetViewportWithCount-viewportCount-03394
`viewportCount` **must** be between 1 and `VkPhysicalDeviceLimits::maxViewports`, inclusive
- VUID-vkCmdSetViewportWithCount-viewportCount-03395
If the `multiViewport` feature is not enabled, `viewportCount` **must** be 1

Valid Usage (Implicit)

- VUID-vkCmdSetViewportWithCount-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetViewportWithCount-pViewports-parameter
`pViewports` **must** be a valid pointer to an array of `viewportCount` valid `VkViewport` structures
- VUID-vkCmdSetViewportWithCount-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetViewportWithCount-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdSetViewportWithCount-viewportCount-arraylength
`viewportCount` **must** be greater than 0

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

To [dynamically set](#) the scissor count and scissor rectangular bounds, call:

```
// Provided by VK_EXT_extended_dynamic_state
void vkCmdSetScissorWithCountEXT(
    VkCommandBuffer          commandBuffer,
    uint32_t                 scissorCount,
    const VkRect2D*          pScissors);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `scissorCount` specifies the scissor count.
- `pScissors` specifies the scissors to use for drawing.

This command sets the scissor count and scissor rectangular bounds state for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the corresponding `VkPipelineViewportStateCreateInfo::scissorCount` and `pScissors` values used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetScissorWithCount-None-08971
At least one of the following **must** be true:
 - the `extendedDynamicState` feature is enabled
- VUID-vkCmdSetScissorWithCount-scissorCount-03397
`scissorCount` **must** be between 1 and `VkPhysicalDeviceLimits::maxViewports`, inclusive
- VUID-vkCmdSetScissorWithCount-scissorCount-03398
If the `multiViewport` feature is not enabled, `scissorCount` **must** be 1
- VUID-vkCmdSetScissorWithCount-x-03399

The `x` and `y` members of `offset` member of any element of `pScissors` **must** be greater than or equal to `0`

- VUID-vkCmdSetScissorWithCount-offset-03400
Evaluation of $(\text{offset.x} + \text{extent.width})$ **must** not cause a signed integer addition overflow for any element of `pScissors`
- VUID-vkCmdSetScissorWithCount-offset-03401
Evaluation of $(\text{offset.y} + \text{extent.height})$ **must** not cause a signed integer addition overflow for any element of `pScissors`

Valid Usage (Implicit)

- VUID-vkCmdSetScissorWithCount-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetScissorWithCount-pScissors-parameter
`pScissors` **must** be a valid pointer to an array of `scissorCount` `VkRect2D` structures
- VUID-vkCmdSetScissorWithCount-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdSetScissorWithCount-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdSetScissorWithCount-scissorCount-arraylength
`scissorCount` **must** be greater than `0`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineViewportStateCreateFlags;
```

`VkPipelineViewportStateCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

A *pre-rasterization shader stage* **can** direct each primitive to one of several viewports. The destination viewport for a primitive is selected by the last active *pre-rasterization shader stage* that has an output variable decorated with `ViewportIndex`. The viewport transform uses the viewport corresponding to the value assigned to `ViewportIndex`, and taken from an implementation-dependent vertex of each primitive. If `ViewportIndex` is outside the range zero to `viewportCount` minus one for a primitive, or if the last active *pre-rasterization shader stage* did not assign a value to `ViewportIndex` for all vertices of a primitive due to flow control, the values resulting from the viewport transformation of the vertices of such primitives are undefined. If the last *pre-rasterization shader stage* does not have an output decorated with `ViewportIndex`, the viewport numbered zero is used by the viewport transformation.

A single vertex **can** be used in more than one individual primitive, in primitives such as `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP`. In this case, the viewport transformation is applied separately for each primitive.

To *dynamically set* the viewport transformation parameters, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetViewport(
    VkCommandBuffer          commandBuffer,
    uint32_t                 firstViewport,
    uint32_t                 viewportCount,
    const VkViewport*        pViewports);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `firstViewport` is the index of the first viewport whose parameters are updated by the command.
- `viewportCount` is the number of viewports whose parameters are updated by the command.
- `pViewports` is a pointer to an array of `VkViewport` structures specifying viewport parameters.

This command sets the viewport transformation parameters state for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_VIEWPORT` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineViewportStateCreateInfo::pViewports` values used to create the currently active pipeline.

The viewport parameters taken from element `i` of `pViewports` replace the current state for the viewport index `firstViewport + i`, for `i` in `[0, viewportCount)`.

Valid Usage

- VUID-vkCmdSetViewport-firstViewport-01223
The sum of `firstViewport` and `viewportCount` **must** be between `1` and `VkPhysicalDeviceLimits::maxViewports`, inclusive
- VUID-vkCmdSetViewport-firstViewport-01224

If the `multiViewport` feature is not enabled, `firstViewport` **must** be 0

- VUID-vkCmdSetViewport-viewportCount-01225

If the `multiViewport` feature is not enabled, `viewportCount` **must** be 1

Valid Usage (Implicit)

- VUID-vkCmdSetViewport-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetViewport-pViewports-parameter
`pViewports` **must** be a valid pointer to an array of `viewportCount` valid `VkViewport` structures
- VUID-vkCmdSetViewport-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetViewport-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdSetViewport-viewportCount-arraylength
`viewportCount` **must** be greater than 0

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

Both `VkPipelineViewportStateCreateInfo` and `vkCmdSetViewport` use `VkViewport` to set the viewport transformation parameters.

The `VkViewport` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkViewport {
    float    x;
    float    y;
    float    width;
    float    height;
    float    minDepth;
    float    maxDepth;
} VkViewport;
```

- **x** and **y** are the viewport's upper left corner (x,y).
- **width** and **height** are the viewport's width and height, respectively.
- **minDepth** and **maxDepth** are the depth range for the viewport.

NOTE Despite their names, **minDepth** **can** be less than, equal to, or greater than **maxDepth**.

The framebuffer depth coordinate z_f **may** be represented using either a fixed-point or floating-point representation. However, a floating-point representation **must** be used if the depth/stencil attachment has a floating-point depth component. If an m-bit fixed-point representation is used, we assume that it represents each value $\frac{k}{2^m-1}$, where $k \in \{ 0, 1, \dots, 2^m-1 \}$, as k (e.g. 1.0 is represented in binary as a string of all ones).

The viewport parameters shown in the above equations are found from these values as

$$O_x = x + \text{width} / 2$$

$$O_y = y + \text{height} / 2$$

$$O_z = \text{minDepth}$$

$$p_x = \text{width}$$

$$p_y = \text{height}$$

$$p_z = \text{maxDepth} - \text{minDepth}$$

The application **can** specify a negative term for **height**, which has the effect of negating the y coordinate in clip space before performing the transform. When using a negative **height**, the application **should** also adjust the **y** value to point to the lower left corner of the viewport instead of the upper left corner. Using the negative **height** allows the application to avoid having to negate the y component of the **Position** output from the last [pre-rasterization shader stage](#).

The width and height of the [implementation-dependent maximum viewport dimensions](#) **must** be greater than or equal to the width and height of the largest image which **can** be created and attached to a framebuffer.

The floating-point viewport bounds are represented with an [implementation-dependent precision](#).

Valid Usage

- VUID-VkViewport-width-01770
width must be greater than **0.0**
- VUID-VkViewport-width-01771
width must be less than or equal to **VkPhysicalDeviceLimits::maxViewportDimensions[0]**
- VUID-VkViewport-height-01773
The absolute value of **height must** be less than or equal to **VkPhysicalDeviceLimits::maxViewportDimensions[1]**
- VUID-VkViewport-x-01774
x must be greater than or equal to **viewportBoundsRange[0]**
- VUID-VkViewport-x-01232
(**x + width**) **must** be less than or equal to **viewportBoundsRange[1]**
- VUID-VkViewport-y-01775
y must be greater than or equal to **viewportBoundsRange[0]**
- VUID-VkViewport-y-01776
y must be less than or equal to **viewportBoundsRange[1]**
- VUID-VkViewport-y-01777
(**y + height**) **must** be greater than or equal to **viewportBoundsRange[0]**
- VUID-VkViewport-y-01233
(**y + height**) **must** be less than or equal to **viewportBoundsRange[1]**
- VUID-VkViewport-minDepth-01234
If the [VK_EXT_depth_range_unrestricted](#) extension is not enabled, **minDepth must** be between **0.0** and **1.0**, inclusive
- VUID-VkViewport-maxDepth-01235
If the [VK_EXT_depth_range_unrestricted](#) extension is not enabled, **maxDepth must** be between **0.0** and **1.0**, inclusive

Chapter 26. Rasterization

Rasterization is the process by which a primitive is converted to a two-dimensional image. Each discrete location of this image contains associated data such as depth, color, or other attributes.

Rasterizing a primitive begins by determining which squares of an integer grid in framebuffer coordinates are occupied by the primitive, and assigning one or more depth values to each such square. This process is described below for points, lines, and polygons.

A grid square, including its (x,y) framebuffer coordinates, z (depth), and associated data added by fragment shaders, is called a fragment. A fragment is located by its upper left corner, which lies on integer grid coordinates.

Rasterization operations also refer to a fragment's sample locations, which are offset by fractional values from its upper left corner. The rasterization rules for points, lines, and triangles involve testing whether each sample location is inside the primitive. Fragments need not actually be square, and rasterization rules are not affected by the aspect ratio of fragments. Display of non-square grids, however, will cause rasterized points and line segments to appear fatter in one direction than the other.

We assume that fragments are square, since it simplifies antialiasing and texturing. After rasterization, fragments are processed by [fragment operations](#).

Several factors affect rasterization, including the members of [VkPipelineRasterizationStateCreateInfo](#) and [VkPipelineMultisampleStateCreateInfo](#).

The [VkPipelineRasterizationStateCreateInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineRasterizationStateCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkPipelineRasterizationStateCreateFlags flags;
    VkBool32                  depthClampEnable;
    VkBool32                  rasterizerDiscardEnable;
    VkPolygonMode              polygonMode;
    VkCullModeFlags            cullMode;
    VkFrontFace                frontFace;
    VkBool32                  depthBiasEnable;
    float                     depthBiasConstantFactor;
    float                     depthBiasClamp;
    float                     depthBiasSlopeFactor;
    float                     lineWidth;
} VkPipelineRasterizationStateCreateInfo;
```

- `sType` is a [VkStructureType](#) value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.

- **depthClampEnable** controls whether to clamp the fragment's depth values as described in [Depth Test](#). If the pipeline is not created with [VkPipelineRasterizationDepthClipStateCreateInfoEXT](#) present then enabling depth clamp will also disable clipping primitives to the z planes of the frustum as described in [Primitive Clipping](#). Otherwise depth clipping is controlled by the state set in [VkPipelineRasterizationDepthClipStateCreateInfoEXT](#).
- **rasterizerDiscardEnable** controls whether primitives are discarded immediately before the rasterization stage.
- **polygonMode** is the triangle rendering mode. See [VkPolygonMode](#).
- **cullMode** is the triangle facing direction used for primitive culling. See [VkCullModeFlagBits](#).
- **frontFace** is a [VkFrontFace](#) value specifying the front-facing triangle orientation to be used for culling.
- **depthBiasEnable** controls whether to bias fragment depth values.
- **depthBiasConstantFactor** is a scalar factor controlling the constant depth value added to each fragment.
- **depthBiasClamp** is the maximum (or minimum) depth bias of a fragment.
- **depthBiasSlopeFactor** is a scalar factor applied to a fragment's slope in depth bias calculations.
- **lineWidth** is the width of rasterized line segments.

Valid Usage

- VUID-VkPipelineRasterizationStateCreateInfo-depthClampEnable-00782
If the **depthClamp** feature is not enabled, **depthClampEnable** **must** be **VK_FALSE**
- VUID-VkPipelineRasterizationStateCreateInfo-polygonMode-01507
If the **fillModeNonSolid** feature is not enabled, **polygonMode** **must** be **VK_POLYGON_MODE_FILL**

Valid Usage (Implicit)

- VUID-VkPipelineRasterizationStateCreateInfo-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO**
- VUID-VkPipelineRasterizationStateCreateInfo-pNext-pNext
Each **pNext** member of any structure (including this one) in the **pNext** chain **must** be either **NULL** or a pointer to a valid instance of [VkPipelineRasterizationConservativeStateCreateInfoEXT](#), [VkPipelineRasterizationDepthClipStateCreateInfoEXT](#), or [VkPipelineRasterizationLineStateCreateInfo](#)
- VUID-VkPipelineRasterizationStateCreateInfo-sType-unique
The **sType** value of each structure in the **pNext** chain **must** be unique
- VUID-VkPipelineRasterizationStateCreateInfo-flags-zeroBitmask
flags **must** be 0
- VUID-VkPipelineRasterizationStateCreateInfo-polygonMode-parameter
polygonMode **must** be a valid [VkPolygonMode](#) value

- VUID-VkPipelineRasterizationStateCreateInfo-cullMode-parameter **cullMode** **must** be a valid combination of [VkCullModeFlagBits](#) values
- VUID-VkPipelineRasterizationStateCreateInfo-frontFace-parameter **frontFace** **must** be a valid [VkFrontFace](#) value

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineRasterizationStateCreateFlags;
```

[VkPipelineRasterizationStateCreateFlags](#) is a bitmask type for setting a mask, but is currently reserved for future use.

If the **pNext** chain of [VkPipelineRasterizationStateCreateInfo](#) includes a [VkPipelineRasterizationDepthClipStateCreateInfoEXT](#) structure, then that structure controls whether depth clipping is enabled or disabled.

The [VkPipelineRasterizationDepthClipStateCreateInfoEXT](#) structure is defined as:

```
// Provided by VK_EXT_depth_clip_enable
typedef struct VkPipelineRasterizationDepthClipStateCreateInfoEXT {
    VkStructureType              sType;
    const void*                  pNext;
    VkPipelineRasterizationDepthClipStateCreateFlagsEXT flags;
    VkBool32                     depthClipEnable;
} VkPipelineRasterizationDepthClipStateCreateInfoEXT;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **flags** is reserved for future use.
- **depthClipEnable** controls whether depth clipping is enabled as described in [Primitive Clipping](#).

Valid Usage (Implicit)

- VUID-VkPipelineRasterizationDepthClipStateCreateInfoEXT-sType-sType **must** be [VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_DEPTH_CLIP_STATE_CREATE_INFO_EXT](#)
- VUID-VkPipelineRasterizationDepthClipStateCreateInfoEXT-flags-zero bitmask **flags** **must** be 0

```
// Provided by VK_EXT_depth_clip_enable
typedef VkFlags VkPipelineRasterizationDepthClipStateCreateFlagsEXT;
```

[VkPipelineRasterizationDepthClipStateCreateFlagsEXT](#) is a bitmask type for setting a mask, but is

currently reserved for future use.

The `VkPipelineMultisampleStateCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineMultisampleStateCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkPipelineMultisampleStateCreateFlags flags;
    VkSampleCountFlagBits    rasterizationSamples;
    VkBool32                 sampleShadingEnable;
    float                    minSampleShading;
    const VkSampleMask*      pSampleMask;
    VkBool32                 alphaToCoverageEnable;
    VkBool32                 alphaToOneEnable;
} VkPipelineMultisampleStateCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `rasterizationSamples` is a `VkSampleCountFlagBits` value specifying the number of samples used in rasterization.
- `sampleShadingEnable` **can** be used to enable [Sample Shading](#).
- `minSampleShading` specifies a minimum fraction of sample shading if `sampleShadingEnable` is `VK_TRUE`.
- `pSampleMask` is a pointer to an array of `VkSampleMask` values used in the [sample mask test](#).
- `alphaToCoverageEnable` controls whether a temporary coverage value is generated based on the alpha component of the fragment's first color output as specified in the [Multisample Coverage](#) section.
- `alphaToOneEnable` controls whether the alpha component of the fragment's first color output is replaced with one as described in [Multisample Coverage](#).

Each bit in the sample mask is associated with a unique [sample index](#) as defined for the [coverage mask](#). Each bit `b` for mask word `w` in the sample mask corresponds to sample index `i`, where $i = 32 \times w + b$. `pSampleMask` has a length equal to $\lceil \text{rasterizationSamples} / 32 \rceil$ words.

If `pSampleMask` is `NULL`, it is treated as if the mask has all bits set to 1.

Valid Usage

- VUID-VkPipelineMultisampleStateCreateInfo-sampleShadingEnable-00784
If the `sampleRateShading` feature is not enabled, `sampleShadingEnable` **must** be `VK_FALSE`
- VUID-VkPipelineMultisampleStateCreateInfo-alphaToOneEnable-00785
If the `alphaToOne` feature is not enabled, `alphaToOneEnable` **must** be `VK_FALSE`

- VUID-VkPipelineMultisampleStateCreateInfo-minSampleShading-00786
`minSampleShading` **must** be in the range [0,1]

Valid Usage (Implicit)

- VUID-VkPipelineMultisampleStateCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO`
- VUID-VkPipelineMultisampleStateCreateInfo-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkPipelineSampleLocationsStateCreateInfoEXT`
- VUID-VkPipelineMultisampleStateCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkPipelineMultisampleStateCreateInfo-flags-zeroBitmask
`flags` **must** be 0
- VUID-VkPipelineMultisampleStateCreateInfo-rasterizationSamples-parameter
`rasterizationSamples` **must** be a valid `VkSampleCountFlagBits` value

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineMultisampleStateCreateFlags;
```

`VkPipelineMultisampleStateCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

The elements of the sample mask array are of type `VkSampleMask`, each representing 32 bits of coverage information:

```
// Provided by VK_VERSION_1_0
typedef uint32_t VkSampleMask;
```

Rasterization only generates fragments which cover one or more pixels inside the framebuffer. Pixels outside the framebuffer are never considered covered in the fragment. Fragments which would be produced by application of any of the primitive rasterization rules described below but which lie outside the framebuffer are not produced, nor are they processed by any later stage of the pipeline, including any of the [fragment operations](#).

Surviving fragments are processed by fragment shaders. Fragment shaders determine associated data for fragments, and **can** also modify or replace their assigned depth values.

26.1. Discarding Primitives Before Rasterization

Primitives are discarded before rasterization if the `rasterizerDiscardEnable` member of `VkPipelineRasterizationStateCreateInfo` is enabled. When enabled, primitives are discarded after they are processed by the last active shader stage in the pipeline before rasterization.

To **dynamically enable** whether primitives are discarded before the rasterization stage, call:

```
// Provided by VK_EXT_extended_dynamic_state2
void vkCmdSetRasterizerDiscardEnableEXT(
    VkCommandBuffer          commandBuffer,
    VkBool32                  rasterizerDiscardEnable);
```

- **commandBuffer** is the command buffer into which the command will be recorded.
- **rasterizerDiscardEnable** controls whether primitives are discarded immediately before the rasterization stage.

This command sets the discard enable for subsequent drawing commands when the graphics pipeline is created with **VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE** set in **VkPipelineDynamicStateCreateInfo::pDynamicStates**. Otherwise, this state is specified by the **VkPipelineRasterizationStateCreateInfo::rasterizerDiscardEnable** value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetRasterizerDiscardEnable-None-08970
At least one of the following **must** be true:
 - the **extendedDynamicState2** feature is enabled

Valid Usage (Implicit)

- VUID-vkCmdSetRasterizerDiscardEnable-commandBuffer-parameter
commandBuffer **must** be a valid **VkCommandBuffer** handle
- VUID-vkCmdSetRasterizerDiscardEnable-commandBuffer-recording
commandBuffer **must** be in the **recording state**
- VUID-vkCmdSetRasterizerDiscardEnable-commandBuffer-cmdpool
The **VkCommandPool** that **commandBuffer** was allocated from **must** support **VK_QUEUE_GRAPHICS_BIT** operations

Host Synchronization

- Host access to **commandBuffer** **must** be externally synchronized
- Host access to the **VkCommandPool** that **commandBuffer** was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

26.2. Rasterization Order

Within a subpass of a [render pass instance](#), for a given (x,y,layer,sample) sample location, the following operations are guaranteed to execute in *rasterization order*, for each separate primitive that includes that sample location:

1. [Fragment operations](#), in the order defined
2. [Blending](#), [logic operations](#), and color writes

Execution of these operations for each primitive in a subpass occurs in [primitive order](#).

26.3. Multisampling

Multisampling is a mechanism to antialias all Vulkan primitives: points, lines, and polygons. The technique is to sample all primitives multiple times at each pixel. Each sample in each framebuffer attachment has storage for a color, depth, and/or stencil value, such that per-fragment operations apply to each sample independently. The color sample values **can** be later *resolved* to a single color (see [Resolving Multisample Images](#) and the [Render Pass](#) chapter for more details on how to resolve multisample images to non-multisample images).

Vulkan defines rasterization rules for single-sample modes in a way that is equivalent to a multisample mode with a single sample in the center of each fragment.

Each fragment includes a [coverage mask](#) with a single bit for each sample in the fragment, and a number of depth values and associated data for each sample.

It is understood that each pixel has [rasterizationSamples](#) locations associated with it. These locations are exact positions, rather than regions or areas, and each is referred to as a sample point. The sample points associated with a pixel **must** be located inside or on the boundary of the unit square that is considered to bound the pixel. Furthermore, the relative locations of sample points **may** be identical for each pixel in the framebuffer, or they **may** differ.

If the current pipeline includes a fragment shader with one or more variables in its interface decorated with [Sample](#) and [Input](#), the data associated with those variables will be assigned independently for each sample. The values for each sample **must** be evaluated at the location of the sample. The data associated with any other variables not decorated with [Sample](#) and [Input](#) need not be evaluated independently for each sample.

A *coverage mask* is generated for each fragment, based on which samples within that fragment are

determined to be within the area of the primitive that generated the fragment.

Single pixel fragments have one set of samples. Multi-pixel fragments defined by setting the [fragment shading rate](#) have one set of samples per pixel. Each set of samples has a number of samples determined by [VkPipelineMultisampleStateCreateInfo::rasterizationSamples](#). Each sample in a set is assigned a unique *sample index* i in the range $[0, \text{rasterizationSamples})$.

Each sample in a fragment is also assigned a unique *coverage index* j in the range $[0, n \times \text{rasterizationSamples})$, where n is the number of sets in the fragment. If the fragment contains a single set of samples, the *coverage index* is always equal to the *sample index*.

If the [fragment shading rate](#) is set, the coverage index j is determined as a function of the *pixel index* p , the *sample index* i , and the number of rasterization samples r as:

$$j = i + r \times ((f_w \times f_h) - 1 - p)$$

where the pixel index p is determined as a function of the pixel's framebuffer location (x,y) and the fragment size (f_w, f_h) :

$$p_x = x \% f_w$$

$$p_y = y \% f_h$$

$$p = p_x + (p_y \times f_w)$$

The tables below illustrate the pixel index for multi-pixel fragments:

Table 29. Pixel Indices - 1 Wide

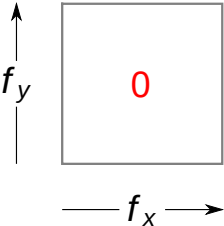
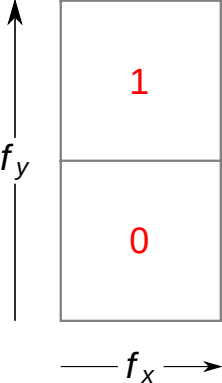
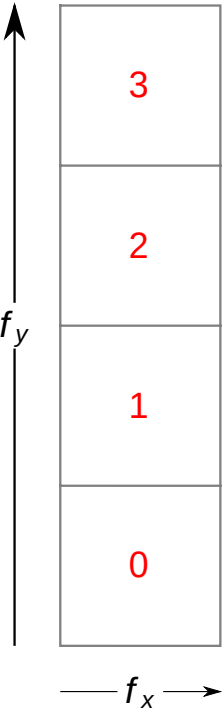
1x1	1x2	1x4
		

Table 30. Pixel Indices - 2 Wide

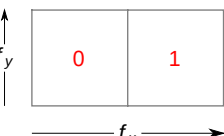
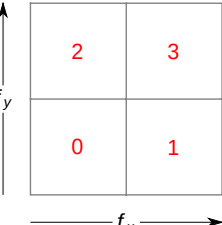
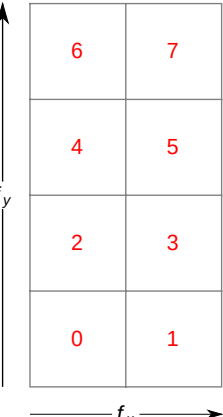
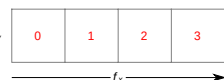
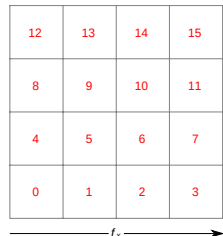
2x1	2x2	2x4
		

Table 31. Pixel Indices - 4 Wide

4x1	4x2	4x4
		

The coverage mask includes B bits packed into W words, defined as:

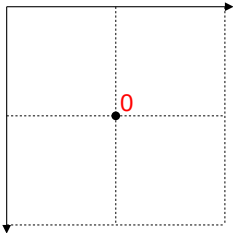
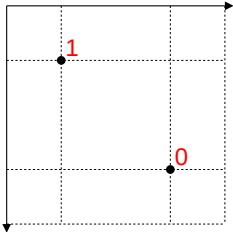
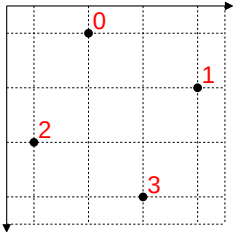
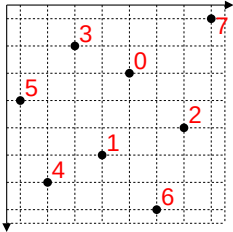
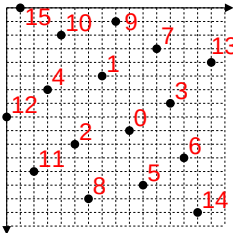
$$B = n \times \text{rasterizationSamples}$$

$$W = \lceil B/32 \rceil$$

Bit b in coverage mask word w is **1** if the sample with coverage index $j = 32 \times w + b$ is covered, and **0** otherwise.

If the `standardSampleLocations` member of `VkPhysicalDeviceLimits` is `VK_TRUE`, then the sample counts `VK_SAMPLE_COUNT_1_BIT`, `VK_SAMPLE_COUNT_2_BIT`, `VK_SAMPLE_COUNT_4_BIT`, `VK_SAMPLE_COUNT_8_BIT`, and `VK_SAMPLE_COUNT_16_BIT` have sample locations as listed in the following table, with the i th entry in the table corresponding to sample index i . `VK_SAMPLE_COUNT_32_BIT` and `VK_SAMPLE_COUNT_64_BIT` do not have standard sample locations. Locations are defined relative to an origin in the upper left corner of the fragment.

Table 32. Standard Sample Locations

Sample count	Sample Locations	
VK_SAMPLE_COUNT_1_BIT	(0.5,0.5)	
VK_SAMPLE_COUNT_2_BIT	(0.75,0.75) (0.25,0.25)	
VK_SAMPLE_COUNT_4_BIT	(0.375, 0.125) (0.875, 0.375) (0.125, 0.625) (0.625, 0.875)	
VK_SAMPLE_COUNT_8_BIT	(0.5625, 0.3125) (0.4375, 0.6875) (0.8125, 0.5625) (0.3125, 0.1875) (0.1875, 0.8125) (0.0625, 0.4375) (0.6875, 0.9375) (0.9375, 0.0625)	
VK_SAMPLE_COUNT_16_BIT	(0.5625, 0.5625) (0.4375, 0.3125) (0.3125, 0.625) (0.75, 0.4375) (0.1875, 0.375) (0.625, 0.8125) (0.8125, 0.6875) (0.6875, 0.1875) (0.375, 0.875) (0.5, 0.0625) (0.25, 0.125) (0.125, 0.75) (0.0, 0.5) (0.9375, 0.25) (0.875, 0.9375) (0.0625, 0.0)	

26.4. Custom Sample Locations

Applications **can** also control the sample locations used for rasterization.

If the `pNext` chain of the `VkPipelineMultisampleStateCreateInfo` structure specified at pipeline creation time includes a `VkPipelineSampleLocationsStateCreateInfoEXT` structure, then that structure controls the sample locations used when rasterizing primitives with the pipeline.

The `VkPipelineSampleLocationsStateCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkPipelineSampleLocationsStateCreateInfoEXT {
    VkStructureType      sType;
    const void*          pNext;
    VkBool32             sampleLocationsEnable;
    VkSampleLocationsInfoEXT sampleLocationsInfo;
} VkPipelineSampleLocationsStateCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `sampleLocationsEnable` controls whether custom sample locations are used. If `sampleLocationsEnable` is `VK_FALSE`, the default sample locations are used and the values specified in `sampleLocationsInfo` are ignored.
- `sampleLocationsInfo` is the sample locations to use during rasterization if `sampleLocationsEnable` is `VK_TRUE` and the graphics pipeline is not created with `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT`.

Valid Usage (Implicit)

- VUID-VkPipelineSampleLocationsStateCreateInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_SAMPLE_LOCATIONS_STATE_CREATE_INFO_EXT`
- VUID-VkPipelineSampleLocationsStateCreateInfoEXT-sampleLocationsInfo-parameter
`sampleLocationsInfo` **must** be a valid `VkSampleLocationsInfoEXT` structure

The `VkSampleLocationsInfoEXT` structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkSampleLocationsInfoEXT {
    VkStructureType      sType;
    const void*          pNext;
    VkSampleCountFlagBits sampleLocationsPerPixel;
    VkExtent2D           sampleLocationGridSize;
    uint32_t             sampleLocationsCount;
    const VkSampleLocationEXT* pSampleLocations;
} VkSampleLocationsInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `sampleLocationsPerPixel` is a `VkSampleCountFlagBits` value specifying the number of sample locations per pixel.
- `sampleLocationGridSize` is the size of the sample location grid to select custom sample locations for.
- `sampleLocationsCount` is the number of sample locations in `pSampleLocations`.
- `pSampleLocations` is a pointer to an array of `sampleLocationsCount` `VkSampleLocationEXT` structures.

This structure **can** be used either to specify the sample locations to be used for rendering or to specify the set of sample locations an image subresource has been last rendered with for the purposes of layout transitions of depth/stencil images created with `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT`.

The sample locations in `pSampleLocations` specify `sampleLocationsPerPixel` number of sample locations for each pixel in the grid of the size specified in `sampleLocationGridSize`. The sample location for sample `i` at the pixel grid location `(x,y)` is taken from `pSampleLocations[(x + y × sampleLocationGridSize.width) × sampleLocationsPerPixel + i]`.

Valid Usage

- VUID-VkSampleLocationsInfoEXT-sampleLocationsPerPixel-01526
`sampleLocationsPerPixel` **must** be a valid `VkSampleCountFlagBits` value that is set in `VkPhysicalDeviceSampleLocationsPropertiesEXT::sampleLocationSampleCounts`
- VUID-VkSampleLocationsInfoEXT-sampleLocationsCount-01527
`sampleLocationsCount` **must** equal `sampleLocationsPerPixel × sampleLocationGridSize.width × sampleLocationGridSize.height`

Valid Usage (Implicit)

- VUID-VkSampleLocationsInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SAMPLE_LOCATIONS_INFO_EXT`
- VUID-VkSampleLocationsInfoEXT-pSampleLocations-parameter
If `sampleLocationsCount` is not 0, `pSampleLocations` **must** be a valid pointer to an array of `sampleLocationsCount` `VkSampleLocationEXT` structures

The `VkSampleLocationEXT` structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkSampleLocationEXT {
    float    x;
    float    y;
} VkSampleLocationEXT;
```

- **x** is the horizontal coordinate of the sample's location.
- **y** is the vertical coordinate of the sample's location.

The domain space of the sample location coordinates has an upper-left origin within the pixel in framebuffer space.

The values specified in a **VkSampleLocationEXT** structure are always clamped to the implementation-dependent sample location coordinate range **[sampleLocationCoordinateRange[0],sampleLocationCoordinateRange[1]]** that **can** be queried using **VkPhysicalDeviceSampleLocationsPropertiesEXT**.

To **dynamically set** the sample locations used for rasterization, call:

```
// Provided by VK_EXT_sample_locations
void vkCmdSetSampleLocationsEXT(
    VkCommandBuffer          commandBuffer,
    const VkSampleLocationsInfoEXT* pSampleLocationsInfo);
```

- **commandBuffer** is the command buffer into which the command will be recorded.
- **pSampleLocationsInfo** is the sample locations state to set.

This command sets the custom sample locations for subsequent drawing commands when the graphics pipeline is created with **VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT** set in **VkPipelineDynamicStateCreateInfo::pDynamicStates**, and when the **VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable** property of the bound graphics pipeline is **VK_TRUE**. Otherwise, this state is specified by the **VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsInfo** values used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetSampleLocationsEXT-variableSampleLocations-01530
If **VkPhysicalDeviceSampleLocationsPropertiesEXT::variableSampleLocations** is **VK_FALSE** then the current render pass **must** have been begun by specifying a **VkRenderPassSampleLocationsBeginInfoEXT** structure whose **pPostSubpassSampleLocations** member contains an element with a **subpassIndex** matching the current subpass index and the **sampleLocationsInfo** member of that element **must** match the sample locations state pointed to by **pSampleLocationsInfo**

Valid Usage (Implicit)

- VUID-vkCmdSetSampleLocationsEXT-commandBuffer-parameter `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetSampleLocationsEXT-pSampleLocationsInfo-parameter `pSampleLocationsInfo` **must** be a valid pointer to a valid `VkSampleLocationsInfoEXT` structure
- VUID-vkCmdSetSampleLocationsEXT-commandBuffer-recording `commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetSampleLocationsEXT-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

26.5. Fragment Shading Rates

The features advertised by `VkPhysicalDeviceFragmentShadingRateFeaturesKHR` allow an application to control the `shading rate` of a given fragment shader invocation.

The fragment shading rate strongly interacts with `Multisampling`, and the set of available rates for an implementation **may** be restricted by sample rate.

To query available shading rates, call:

```
// Provided by VK_KHR_fragment_shading_rate
VkResult vkGetPhysicalDeviceFragmentShadingRatesKHR(
    VkPhysicalDevice          physicalDevice,
    uint32_t*                 pFragmentShadingRateCount,
    VkPhysicalDeviceFragmentShadingRateKHR* pFragmentShadingRates);
```

- `physicalDevice` is the handle to the physical device whose properties will be queried.
- `pFragmentShadingRateCount` is a pointer to an integer related to the number of fragment shading rates available or queried, as described below.
- `pFragmentShadingRates` is either `NULL` or a pointer to an array of `VkPhysicalDeviceFragmentShadingRateKHR` structures.

If `pFragmentShadingRates` is `NULL`, then the number of fragment shading rates available is returned in `pFragmentShadingRateCount`. Otherwise, `pFragmentShadingRateCount` **must** point to a variable set by the application to the number of elements in the `pFragmentShadingRates` array, and on return the variable is overwritten with the number of structures actually written to `pFragmentShadingRates`. If `pFragmentShadingRateCount` is less than the number of fragment shading rates available, at most `pFragmentShadingRateCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available fragment shading rates were returned.

The returned array of fragment shading rates **must** be ordered from largest `fragmentSize.width` value to smallest, and each set of fragment shading rates with the same `fragmentSize.width` value **must** be ordered from largest `fragmentSize.height` to smallest. Any two entries in the array **must** not have the same `fragmentSize` values.

For any entry in the array, the following rules also apply:

- The value of `fragmentSize.width` **must** be less than or equal to `maxFragmentSize.width`.
- The value of `fragmentSize.width` **must** be greater than or equal to 1.
- The value of `fragmentSize.width` **must** be a power-of-two.
- The value of `fragmentSize.height` **must** be less than or equal to `maxFragmentSize.height`.
- The value of `fragmentSize.height` **must** be greater than or equal to 1.
- The value of `fragmentSize.height` **must** be a power-of-two.
- The highest sample count in `sampleCounts` **must** be less than or equal to `maxFragmentShadingRateRasterizationSamples`.
- The product of `fragmentSize.width`, `fragmentSize.height`, and the highest sample count in `sampleCounts` **must** be less than or equal to `maxFragmentShadingRateCoverageSamples`.

Implementations **must** support at least the following shading rates:

<code>sampleCounts</code>	<code>fragmentSize</code>
<code>VK_SAMPLE_COUNT_1_BIT VK_SAMPLE_COUNT_4_BIT</code>	{2,2}
<code>VK_SAMPLE_COUNT_1_BIT VK_SAMPLE_COUNT_4_BIT</code>	{2,1}
<code>~0</code>	{1,1}

If `framebufferColorSampleCounts`, includes `VK_SAMPLE_COUNT_2_BIT`, the required rates **must** also include `VK_SAMPLE_COUNT_2_BIT`.

NOTE

Including the {1,1} fragment size is done for completeness; it has no actual effect on the support of rendering without setting the fragment size. All sample counts are

supported for this rate.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDeviceFragmentShadingRatesKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceFragmentShadingRatesKHR-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceFragmentShadingRatesKHR-pFragmentShadingRateCount-parameter `pFragmentShadingRateCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceFragmentShadingRatesKHR-pFragmentShadingRates-parameter

If the value referenced by `pFragmentShadingRateCount` is not `0`, and `pFragmentShadingRates` is not `NULL`, `pFragmentShadingRates` **must** be a valid pointer to an array of `pFragmentShadingRateCount` `VkPhysicalDeviceFragmentShadingRateKHR` structures

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkPhysicalDeviceFragmentShadingRateKHR` structure is defined as

```
// Provided by VK_KHR_fragment_shading_rate
typedef struct VkPhysicalDeviceFragmentShadingRateKHR {
    VkStructureType    sType;
    void*              pNext;
    VkSampleCountFlags sampleCounts;
    VkExtent2D         fragmentSize;
} VkPhysicalDeviceFragmentShadingRateKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `sampleCounts` is a bitmask of sample counts for which the shading rate described by `fragmentSize` is supported.

- `fragmentSize` is a `VkExtent2D` describing the width and height of a supported shading rate.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentShadingRateKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_KHR`
- VUID-VkPhysicalDeviceFragmentShadingRateKHR-pNext-pNext
`pNext` **must** be `NULL`

Fragment shading rates **can** be set at three points, with the three rates combined to determine the final shading rate.

26.5.1. Pipeline Fragment Shading Rate

The *pipeline fragment shading rate* **can** be set on a per-draw basis by either setting the rate in a graphics pipeline, or dynamically via `vkCmdSetFragmentShadingRateKHR`.

The `VkPipelineFragmentShadingRateStateCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_fragment_shading_rate
typedef struct VkPipelineFragmentShadingRateStateCreateInfoKHR {
    VkStructureType          sType;
    const void*              pNext;
    VkExtent2D               fragmentSize;
    VkFragmentShadingRateCombinerOpKHR    combinerOps[2];
} VkPipelineFragmentShadingRateStateCreateInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `fragmentSize` specifies a `VkExtent2D` structure containing the fragment size used to define the pipeline fragment shading rate for drawing commands using this pipeline.
- `combinerOps` specifies a `VkFragmentShadingRateCombinerOpKHR` value determining how the `pipeline`, `primitive`, and `attachment shading rates` are `combined` for fragments generated by drawing commands using the created pipeline.

If the `pNext` chain of `VkGraphicsPipelineCreateInfo` includes a `VkPipelineFragmentShadingRateStateCreateInfoKHR` structure, then that structure includes parameters controlling the pipeline fragment shading rate.

If this structure is not present, `fragmentSize` is considered to be equal to (1,1), and both elements of `combinerOps` are considered to be equal to `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR`.

Valid Usage (Implicit)

- VUID-VkPipelineFragmentShadingRateStateCreateInfoKHR-sType-sType

`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_FRAGMENT_SHADING_RATE_STATE_CREATE_INFO_KHR`

To [dynamically set](#) the pipeline fragment shading rate and combiner operation, call:

```
// Provided by VK_KHR_fragment_shading_rate
void vkCmdSetFragmentShadingRateKHR(
    VkCommandBuffer          commandBuffer,
    const VkExtent2D*        pFragmentSize,
    const VkFragmentShadingRateCombinerOpKHR combinerOps[2]);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pFragmentSize` specifies the pipeline fragment shading rate for subsequent drawing commands.
- `combinerOps` specifies a `VkFragmentShadingRateCombinerOpKHR` determining how the [pipeline](#), [primitive](#), and [attachment shading rates](#) are [combined](#) for fragments generated by subsequent drawing commands.

This command sets the pipeline fragment shading rate and combiner operation for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineFragmentShadingRateStateCreateInfoKHR` values used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetFragmentShadingRateKHR-pipelineFragmentShadingRate-04507
If the `pipelineFragmentShadingRate` feature is not enabled, `pFragmentSize->width` **must** be 1
- VUID-vkCmdSetFragmentShadingRateKHR-pipelineFragmentShadingRate-04508
If the `pipelineFragmentShadingRate` feature is not enabled, `pFragmentSize->height` **must** be 1
- VUID-vkCmdSetFragmentShadingRateKHR-pipelineFragmentShadingRate-04509
One of the `pipelineFragmentShadingRate`, `primitiveFragmentShadingRate`, or `attachmentFragmentShadingRate` features **must** be enabled
- VUID-vkCmdSetFragmentShadingRateKHR-primitiveFragmentShadingRate-04510
If the `primitiveFragmentShadingRate` feature is not enabled, `combinerOps[0]` **must** be `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR`
- VUID-vkCmdSetFragmentShadingRateKHR-attachmentFragmentShadingRate-04511
If the `attachmentFragmentShadingRate` feature is not enabled, `combinerOps[1]` **must** be `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR`
- VUID-vkCmdSetFragmentShadingRateKHR-fragmentSizeNonTrivialCombinerOps-04512
If the `fragmentSizeNonTrivialCombinerOps` limit is not supported, elements of `combinerOps` **must** be either `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR` or `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_REPLACE_KHR`
- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-04513

pFragmentSize->width **must** be greater than or equal to 1

- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-04514
pFragmentSize->height **must** be greater than or equal to 1
- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-04515
pFragmentSize->width **must** be a power-of-two value
- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-04516
pFragmentSize->height **must** be a power-of-two value
- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-04517
pFragmentSize->width **must** be less than or equal to 4
- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-04518
pFragmentSize->height **must** be less than or equal to 4

Valid Usage (Implicit)

- VUID-vkCmdSetFragmentShadingRateKHR-commandBuffer-parameter
commandBuffer **must** be a valid [VkCommandBuffer](#) handle
- VUID-vkCmdSetFragmentShadingRateKHR-pFragmentSize-parameter
pFragmentSize **must** be a valid pointer to a valid [VkExtent2D](#) structure
- VUID-vkCmdSetFragmentShadingRateKHR-combinerOps-parameter
Each element of combinerOps **must** be a valid [VkFragmentShadingRateCombinerOpKHR](#) value
- VUID-vkCmdSetFragmentShadingRateKHR-commandBuffer-recording
commandBuffer **must** be in the [recording state](#)
- VUID-vkCmdSetFragmentShadingRateKHR-commandBuffer-cmdpool
The [VkCommandPool](#) that commandBuffer was allocated from **must** support [VK_QUEUE_GRAPHICS_BIT](#) operations

Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the [VkCommandPool](#) that commandBuffer was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

26.5.2. Primitive Fragment Shading Rate

The *primitive fragment shading rate* **can** be set via the `PrimitiveShadingRateKHR` built-in in the last active `pre-rasterization shader stage`. The rate associated with a given primitive is sourced from the value written to `PrimitiveShadingRateKHR` by that primitive's `provoking vertex`.

26.5.3. Attachment Fragment Shading Rate

The *attachment shading rate* **can** be set by including `VkFragmentShadingRateAttachmentInfoKHR` in a subpass to define a *fragment shading rate attachment*. Each pixel in the framebuffer is assigned an attachment fragment shading rate by the corresponding texel in the fragment shading rate attachment, according to:

$$x' = \text{floor}(x / \text{region}_x)$$

$$y' = \text{floor}(y / \text{region}_y)$$

where x' and y' are the coordinates of a texel in the fragment shading rate attachment, x and y are the coordinates of the pixel in the framebuffer, and region_x and region_y are the size of the region each texel corresponds to, as defined by the `shadingRateAttachmentTexelSize` member of `VkFragmentShadingRateAttachmentInfoKHR`.

If `multiview is enabled` and the shading rate attachment has multiple layers, the shading rate attachment texel is selected using `layer = ViewIndex`. If `multiview is disabled`, and both the shading rate attachment and the framebuffer have multiple layers, the shading rate attachment texel is selected using `layer = Layer`. Otherwise, `layer = 0`.

The texel is `read from the fragment shading rate attachment image` using image coordinates equal to $(x', y', 0, \text{layer}, 0, 0)$. The fragment size is encoded into the first component of the result of that operation as follows:

$$\text{size}_w = 2^{((\text{texel}/4)\&3)}$$

$$\text{size}_h = 2^{(\text{texel}\&3)}$$

where `texel` is the value in the first component of the returned value, and size_w and size_h are the width and height of the fragment size, decoded from the texel.

If no fragment shading rate attachment is specified, this size is calculated as $\text{size}_w = \text{size}_h = 1$. Applications **must** not specify a width or height greater than 4 by this method.

The *Fragment Shading Rate* enumeration in SPIR-V adheres to the above encoding.

26.5.4. Combining the Fragment Shading Rates

The final rate (C_{xy}) used for fragment shading **must** be one of the rates returned by `vkGetPhysicalDeviceFragmentShadingRatesKHR` for the sample count used by rasterization.

If any of the following conditions are met, C_{xy} is set to {1,1} by the implementation:

- If `Sample Shading` is enabled.
- The `fragmentShadingRateWithSampleMask` limit is not supported, and `VkPipelineMultisampleStateCreateInfo::pSampleMask` contains a zero value in any bit used by fragment operations.
- The `fragmentShadingRateWithShaderSampleMask` is not supported, and the fragment shader has `SampleMask` in the input or output interface.
- The `fragmentShadingRateWithShaderDepthStencilWrites` limit is not supported, and the fragment shader declares the `FragDepth` or `FragStencilRefEXT` built-in.
- The `fragmentShadingRateWithConservativeRasterization` limit is not supported, and `VkPipelineRasterizationConservativeStateCreateInfoEXT::conservativeRasterizationMode` is not `VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT`.
- The `fragmentShadingRateWithFragmentShaderInterlock` limit is not supported, and the fragment shader declares any of the `fragment shader interlock` execution modes.
- The `fragmentShadingRateWithCustomSampleLocations` limit is not supported, and `VkPipelineSampleLocationsStateCreateInfoEXT::sampleLocationsEnable` is `VK_TRUE`.

Otherwise, each of the specified shading rates are combined and then used to derive the value of C_{xy} . As there are three ways to specify shading rates, two combiner operations are specified - between the `pipeline` and `primitive` shading rates, and between the result of that and the `attachment shading rate`.

The equation used for each combiner operation is defined by `VkFragmentShadingRateCombinerOpKHR`:

```
// Provided by VK_KHR_fragment_shading_rate
typedef enum VkFragmentShadingRateCombinerOpKHR {
    VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR = 0,
    VK_FRAGMENT_SHADING_RATE_COMBINER_OP_REPLACE_KHR = 1,
    VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MIN_KHR = 2,
    VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MAX_KHR = 3,
    VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MUL_KHR = 4,
} VkFragmentShadingRateCombinerOpKHR;
```

- `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR` specifies a combiner operation of $\text{combine}(A_{xy}, B_{xy}) = A_{xy}$.
- `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_REPLACE_KHR` specifies a combiner operation of $\text{combine}(A_{xy}, B_{xy}) = B_{xy}$.
- `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MIN_KHR` specifies a combiner operation of $\text{combine}(A_{xy}, B_{xy}) = \min(A_{xy}, B_{xy})$.

- `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MAX_KHR` specifies a combiner operation of $\text{combine}(A_{xy}, B_{xy}) = \max(A_{xy}, B_{xy})$.
- `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MUL_KHR` specifies a combiner operation of $\text{combine}(A_{xy}, B_{xy}) = A_{xy} * B_{xy}$.

where $\text{combine}(A_{xy}, B_{xy})$ is the combine operation, and A_{xy} and B_{xy} are the inputs to the operation.

If `fragmentShadingRateStrictMultiplyCombiner` is `VK_FALSE`, using `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MUL_KHR` with values of 1 for both A and B in the same dimension results in the value 2 being produced for that dimension. See the definition of `fragmentShadingRateStrictMultiplyCombiner` for more information.

These operations are performed in a component-wise fashion.

This is used to generate a combined fragment area using the equation:

$$C_{xy} = \text{combine}(A_{xy}, B_{xy})$$

where C_{xy} is the combined fragment area result, and A_{xy} and B_{xy} are the fragment areas of the fragment shading rates being combined.

Two combine operations are performed, first with A_{xy} equal to the [pipeline fragment shading rate](#) and B_{xy} equal to the [primitive fragment shading rate](#), with the combine() operation selected by `combinerOps[0]`. A second combination is then performed, with A_{xy} equal to the result of the first combination and B_{xy} equal to the [attachment fragment shading rate](#), with the combine() operation selected by `combinerOps[1]`. The result of the second combination is used as the final fragment shading rate, reported via the [ShadingRateKHR built-in](#).

Implementations **should** clamp the inputs to the combiner operations A_{xy} and B_{xy} , and **must** clamp the result of the second combiner operation.

A fragment shading rate R_{xy} representing any of A_{xy} , B_{xy} or C_{xy} is clamped as follows. If R_{xy} is one of the rates returned by [vkGetPhysicalDeviceFragmentShadingRatesKHR](#) for the sample count used by rasterization, the clamped shading rate R_{xy}' is R_{xy} . Otherwise, the clamped shading rate is selected from the rates returned by [vkGetPhysicalDeviceFragmentShadingRatesKHR](#) for the sample count used by rasterization. From this list of supported rates, the following steps are applied in order, to select a single value:

1. Keep only rates where $R_x' \leq R_x$ and $R_y' \leq R_y$.
 - Implementations **may** also keep rates where $R_x' \leq R_y$ and $R_y' \leq R_x$.
2. Keep only rates with the highest area ($R_x' \times R_y'$).
3. Keep only rates with the lowest aspect ratio ($R_x' + R_y'$).
4. In cases where a wide (e.g. 4x1) and tall (e.g. 1x4) rate remain, the implementation **may** choose either rate. However, it **must** choose this rate consistently for the same shading rates, and combiner operations for the lifetime of the [VkDevice](#).

26.6. Sample Shading

Sample shading **can** be used to specify a minimum number of unique samples to process for each fragment. If sample shading is enabled, an implementation **must** invoke the fragment shader at least $\max(\text{VkPipelineMultisampleStateCreateInfo::minSampleShading} \times \text{VkPipelineMultisampleStateCreateInfo::rasterizationSamples}, 1)$ times per fragment. If `VkPipelineMultisampleStateCreateInfo::sampleShadingEnable` is `VK_TRUE`, sample shading is enabled.

If a fragment shader entry point **statically uses** an input variable decorated with a `BuiltIn` of `SampleId` or `SamplePosition`, sample shading is enabled and a value of `1.0` is used instead of `minSampleShading`. If a fragment shader entry point **statically uses** an input variable decorated with `Sample`, sample shading **may** be enabled and a value of `1.0` will be used instead of `minSampleShading` if it is.

NOTE

If a shader decorates an input variable with `Sample` and that value meaningfully impacts the output of a shader, sample shading will be enabled to ensure that the input is in fact interpolated per-sample. This is inherent to the specification and not spelled out here - if an application simply declares such a variable it is implementation-defined whether sample shading is enabled or not. It is possible to see the effects of this by using atomics in the shader or using a pipeline statistics query to query the number of fragment invocations, even if the shader itself does not use any per-sample variables.

If there are fewer fragment invocations than **covered samples**, implementations **may** include those samples in fragment shader invocations in any manner as long as covered samples are all shaded at least once, and each invocation that is not a **helper invocation** covers at least one sample.

26.7. Points

A point is drawn by generating a set of fragments in the shape of a square centered around the vertex of the point. Each vertex has an associated point size controlling the width/height of that square. The point size is taken from the (potentially clipped) shader built-in `PointSize` written by:

- the geometry shader, if active;
- the tessellation evaluation shader, if active and no geometry shader is active;
- the vertex shader, otherwise

and clamped to the implementation-dependent point size range `[pointSizeRange[0], pointSizeRange[1]]`. The value written to `PointSize` **must** be greater than zero.

Not all point sizes need be supported, but the size `1.0` **must** be supported. The range of supported sizes and the size of evenly-spaced gradations within that range are implementation-dependent. The range and gradations are obtained from the `pointSizeRange` and `pointSizeGranularity` members of `VkPhysicalDeviceLimits`. If, for instance, the size range is from `0.1` to `2.0` and the gradation size is `0.1`, then the sizes `0.1`, `0.2`, ..., `1.9`, `2.0` are supported. Additional point sizes **may** also be supported. There is no requirement that these sizes be equally spaced. If an unsupported size is requested, the nearest supported size is used instead.

26.7.1. Basic Point Rasterization

Point rasterization produces a fragment for each fragment area group of framebuffer pixels with one or more sample points that intersect a region centered at the point's (x_p, y_p) . This region is a square with side equal to the current point size. Coverage bits that correspond to sample points that intersect the region are 1, other coverage bits are 0. All fragments produced in rasterizing a point are assigned the same associated data, which are those of the vertex corresponding to the point. However, the fragment shader built-in `PointCoord` contains point sprite texture coordinates. The s and t point sprite texture coordinates vary from zero to one across the point horizontally left-to-right and vertically top-to-bottom, respectively. The following formulas are used to evaluate s and t :

$$s = \frac{1}{2} + \frac{\tilde{A}x_p - x_f!}{\text{size}}$$
$$t = \frac{1}{2} + \frac{\tilde{A}y_p - y_f!}{\text{size}}$$

where size is the point's size; (x_p, y_p) is the location at which the point sprite coordinates are evaluated - this **may** be the framebuffer coordinates of the fragment center, or the location of a sample; and (x_f, y_f) is the exact, unrounded framebuffer coordinate of the vertex for the point.

26.8. Line Segments

Line segment rasterization options are controlled by the `VkPipelineRasterizationLineStateCreateInfo` structure.

The `VkPipelineRasterizationLineStateCreateInfo` structure is defined as:

```
typedef struct VkPipelineRasterizationLineStateCreateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkLineRasterizationMode    lineRasterizationMode;
    VkBool32            stippledLineEnable;
    uint32_t            lineStippleFactor;
    uint16_t            lineStipplePattern;
} VkPipelineRasterizationLineStateCreateInfo;
```

```
// Provided by VK_KHR_line_rasterization
typedef VkPipelineRasterizationLineStateCreateInfo
VkPipelineRasterizationLineStateCreateInfoKHR;
```

```
// Provided by VK_EXT_line_rasterization
typedef VkPipelineRasterizationLineStateCreateInfo
VkPipelineRasterizationLineStateCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.

- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `lineRasterizationMode` is a `VkLineRasterizationMode` value selecting the style of line rasterization.
- `stippledLineEnable` enables `stippled line rasterization`.
- `lineStippleFactor` is the repeat factor used in stippled line rasterization.
- `lineStipplePattern` is the bit pattern used in stippled line rasterization.

If `stippledLineEnable` is `VK_FALSE`, the values of `lineStippleFactor` and `lineStipplePattern` are ignored.

Valid Usage

- VUID-VkPipelineRasterizationLineStateCreateInfo-lineRasterizationMode-02768
If `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR`, then the `rectangularLines` feature **must** be enabled
- VUID-VkPipelineRasterizationLineStateCreateInfo-lineRasterizationMode-02769
If `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_BRESENHAM`, then the `bresenhamLines` feature **must** be enabled
- VUID-VkPipelineRasterizationLineStateCreateInfo-lineRasterizationMode-02770
If `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH`, then the `smoothLines` feature **must** be enabled
- VUID-VkPipelineRasterizationLineStateCreateInfo-stippledLineEnable-02771
If `stippledLineEnable` is `VK_TRUE` and `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR`, then the `stippledRectangularLines` feature **must** be enabled
- VUID-VkPipelineRasterizationLineStateCreateInfo-stippledLineEnable-02772
If `stippledLineEnable` is `VK_TRUE` and `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_BRESENHAM`, then the `stippledBresenhamLines` feature **must** be enabled
- VUID-VkPipelineRasterizationLineStateCreateInfo-stippledLineEnable-02773
If `stippledLineEnable` is `VK_TRUE` and `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH`, then the `stippledSmoothLines` feature **must** be enabled
- VUID-VkPipelineRasterizationLineStateCreateInfo-stippledLineEnable-02774
If `stippledLineEnable` is `VK_TRUE` and `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_DEFAULT`, then the `stippledRectangularLines` feature **must** be enabled and `VkPhysicalDeviceLimits::strictLines` **must** be `VK_TRUE`

Valid Usage (Implicit)

- VUID-VkPipelineRasterizationLineStateCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO`

- VUID-VkPipelineRasterizationLineStateCreateInfo-lineRasterizationMode-parameter **lineRasterizationMode** **must** be a valid [VkLineRasterizationMode](#) value

Possible values of [VkPipelineRasterizationLineStateCreateInfo::lineRasterizationMode](#), specifying the line rasterization mode, are:

```
typedef enum VkLineRasterizationMode {
    VK_LINE_RASTERIZATION_MODE_DEFAULT = 0,
    VK_LINE_RASTERIZATION_MODE_RECTANGULAR = 1,
    VK_LINE_RASTERIZATION_MODE_BRESENHAM = 2,
    VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH = 3,
    // Provided by VK_EXT_line_rasterization
    VK_LINE_RASTERIZATION_MODE_DEFAULT_EXT = VK_LINE_RASTERIZATION_MODE_DEFAULT,
    // Provided by VK_EXT_line_rasterization
    VK_LINE_RASTERIZATION_MODE_RECTANGULAR_EXT =
VK_LINE_RASTERIZATION_MODE_RECTANGULAR,
    // Provided by VK_EXT_line_rasterization
    VK_LINE_RASTERIZATION_MODE_BRESENHAM_EXT = VK_LINE_RASTERIZATION_MODE_BRESENHAM,
    // Provided by VK_EXT_line_rasterization
    VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_EXT =
VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH,
    // Provided by VK_KHR_line_rasterization
    VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR = VK_LINE_RASTERIZATION_MODE_DEFAULT,
    // Provided by VK_KHR_line_rasterization
    VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR =
VK_LINE_RASTERIZATION_MODE_RECTANGULAR,
    // Provided by VK_KHR_line_rasterization
    VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR = VK_LINE_RASTERIZATION_MODE_BRESENHAM,
    // Provided by VK_KHR_line_rasterization
    VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR =
VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH,
} VkLineRasterizationMode;
```

```
// Provided by VK_KHR_line_rasterization
typedef VkLineRasterizationMode VkLineRasterizationModeKHR;
```

```
// Provided by VK_EXT_line_rasterization
typedef VkLineRasterizationMode VkLineRasterizationModeEXT;
```

- **VK_LINE_RASTERIZATION_MODE_DEFAULT** is equivalent to **VK_LINE_RASTERIZATION_MODE_RECTANGULAR** if [VkPhysicalDeviceLimits::strictLines](#) is **VK_TRUE**, otherwise lines are drawn as non-**strictLines** parallelograms. Both of these modes are defined in [Basic Line Segment Rasterization](#).
- **VK_LINE_RASTERIZATION_MODE_RECTANGULAR** specifies lines drawn as if they were rectangles extruded from the line
- **VK_LINE_RASTERIZATION_MODE_BRESENHAM** specifies lines drawn by determining which pixel

diamonds the line intersects and exits, as defined in [Bresenham Line Segment Rasterization](#).

- `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH` specifies lines drawn if they were rectangles extruded from the line, with alpha falloff, as defined in [Smooth Lines](#).

To [dynamically set](#) the line width, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetLineWidth(
    VkCommandBuffer          commandBuffer,
    float                    lineWidth);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `lineWidth` is the width of rasterized line segments.

This command sets the line width for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_LINE_WIDTH` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineRasterizationStateCreateInfo::lineWidth` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetLineWidth-lineWidth-00788
If the `wideLines` feature is not enabled, `lineWidth` **must** be `1.0`

Valid Usage (Implicit)

- VUID-vkCmdSetLineWidth-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetLineWidth-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetLineWidth-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHIC S_BIT	State

Not all line widths need be supported for line segment rasterization, but width 1.0 antialiased segments **must** be provided. The range and gradations are obtained from the `lineWidthRange` and `lineWidthGranularity` members of `VkPhysicalDeviceLimits`. If, for instance, the size range is from 0.1 to 2.0 and the gradation size is 0.1, then the sizes 0.1, 0.2, ..., 1.9, 2.0 are supported. Additional line widths **may** also be supported. There is no requirement that these widths be equally spaced. If an unsupported width is requested, the nearest supported width is used instead.

26.8.1. Basic Line Segment Rasterization

If the `lineRasterizationMode` member of `VkPipelineRasterizationLineStateCreateInfo` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR`, rasterized line segments produce fragments which intersect a rectangle centered on the line segment. Two of the edges are parallel to the specified line segment; each is at a distance of one-half the current width from that segment in directions perpendicular to the direction of the line. The other two edges pass through the line endpoints and are perpendicular to the direction of the specified line segment. Coverage bits that correspond to sample points that intersect the rectangle are 1, other coverage bits are 0.

Next we specify how the data associated with each rasterized fragment are obtained. Let $\mathbf{p}_r = (x_d, y_d)$ be the framebuffer coordinates at which associated data are evaluated. This **may** be the center of a fragment or the location of a sample within the fragment. When `rasterizationSamples` is `VK_SAMPLE_COUNT_1_BIT`, the fragment center **must** be used. Let $\mathbf{p}_a = (x_a, y_a)$ and $\mathbf{p}_b = (x_b, y_b)$ be initial and final endpoints of the line segment, respectively. Set

$$t = \frac{(\mathbf{p}_r - \mathbf{p}_a) \cdot (\mathbf{p}_b - \mathbf{p}_a)}{\|\mathbf{p}_b - \mathbf{p}_a\|^2}$$

(Note that $t = 0$ at $\mathbf{p}_a$ and $t = 1$ at $\mathbf{p}_b$. Also note that this calculation projects the vector from $\mathbf{p}_a$ to $\mathbf{p}_r$ onto the line, and thus computes the normalized distance of the fragment along the line.)

If `strictLines` is `VK_TRUE`, line segments are rasterized using perspective or linear interpolation.

Perspective interpolation for a line segment interpolates two values in a manner that is correct when taking the perspective of the viewport into consideration, by way of the line segment's clip coordinates. An interpolated value f can be determined by

$$f = \frac{(1 - t)f_a / w_a + tf_b / w_b}{(1 - t) / w_a + t / w_b}$$

where f_a and f_b are the data associated with the starting and ending endpoints of the segment, respectively; w_a and w_b are the clip w coordinates of the starting and ending endpoints of the segment, respectively.

Linear interpolation for a line segment directly interpolates two values, and an interpolated value f can be determined by

$$f = (1 - t) f_a + t f_b$$

where f_a and f_b are the data associated with the starting and ending endpoints of the segment, respectively.

The clip coordinate w for a sample is determined using perspective interpolation. The depth value z for a sample is determined using linear interpolation. Interpolation of fragment shader input values are determined by [Interpolation decorations](#).

The above description documents the preferred method of line rasterization, and **must** be used when `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR`.

When `strictLines` is `VK_FALSE`, and when the `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_DEFAULT`, the edges of the lines are generated as a parallelogram surrounding the original line. The major axis is chosen by noting the axis in which there is the greatest distance between the line start and end points. If the difference is equal in both directions then the X axis is chosen as the major axis. Edges 2 and 3 are aligned to the minor axis and are centered on the endpoints of the line as in [Non strict lines](#), and each is `lineWidth` long. Edges 0 and 1 are parallel to the line and connect the endpoints of edges 2 and 3. Coverage bits that correspond to sample points that intersect the parallelogram are 1, other coverage bits are 0.

Samples that fall exactly on the edge of the parallelogram follow the polygon rasterization rules.

Interpolation occurs as if the parallelogram was decomposed into two triangles where each pair of vertices at each end of the line has identical attributes.

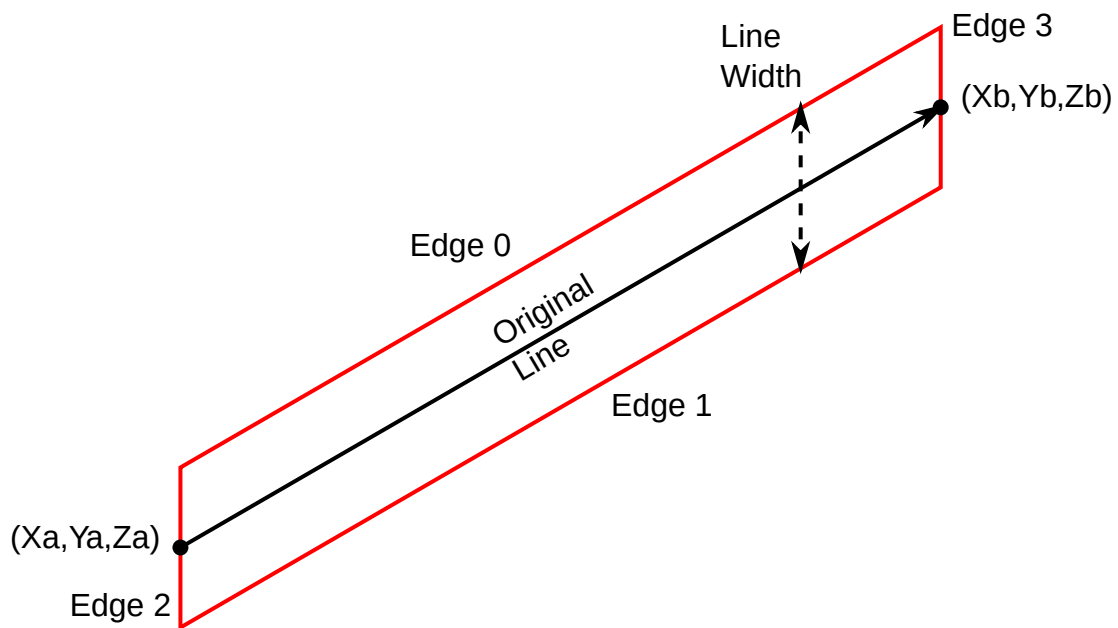


Figure 15. Non strict lines

Only when `strictLines` is `VK_FALSE` implementations **may** deviate from the non-strict line algorithm described above in the following ways:

- Implementations **may** instead interpolate each fragment according to the formula in [Basic Line Segment Rasterization](#) using the original line segment endpoints.
- Rasterization of non-antialiased non-strict line segments **may** be performed using the rules defined in [Bresenham Line Segment Rasterization](#).

26.8.2. Bresenham Line Segment Rasterization

If `lineRasterizationMode` is `VK_LINE_RASTERIZATION_MODE_BRESENHAM`, then the following rules replace the line rasterization rules defined in [Basic Line Segment Rasterization](#).

Non-strict lines **may** also follow these rasterization rules for non-antialiased lines.

Line segment rasterization begins by characterizing the segment as either *x-major* or *y-major*. *x-major* line segments have slope in the closed interval $[-1,1]$; all other line segments are *y-major* (slope is determined by the segment's endpoints). We specify rasterization only for *x-major* segments except in cases where the modifications for *y-major* segments are not self-evident.

Ideally, Vulkan uses a *diamond-exit* rule to determine those fragments that are produced by rasterizing a line segment. For each fragment f with center at framebuffer coordinates x_f and y_f , define a diamond-shaped region that is the intersection of four half planes:

$$R_f = \{(x, y) \mid |x - x_f| + |y - y_f| < \frac{1}{2}\}$$

Essentially, a line segment starting at p_a and ending at p_b produces those fragments f for which the segment intersects R_f , except if p_b is contained in R_f .

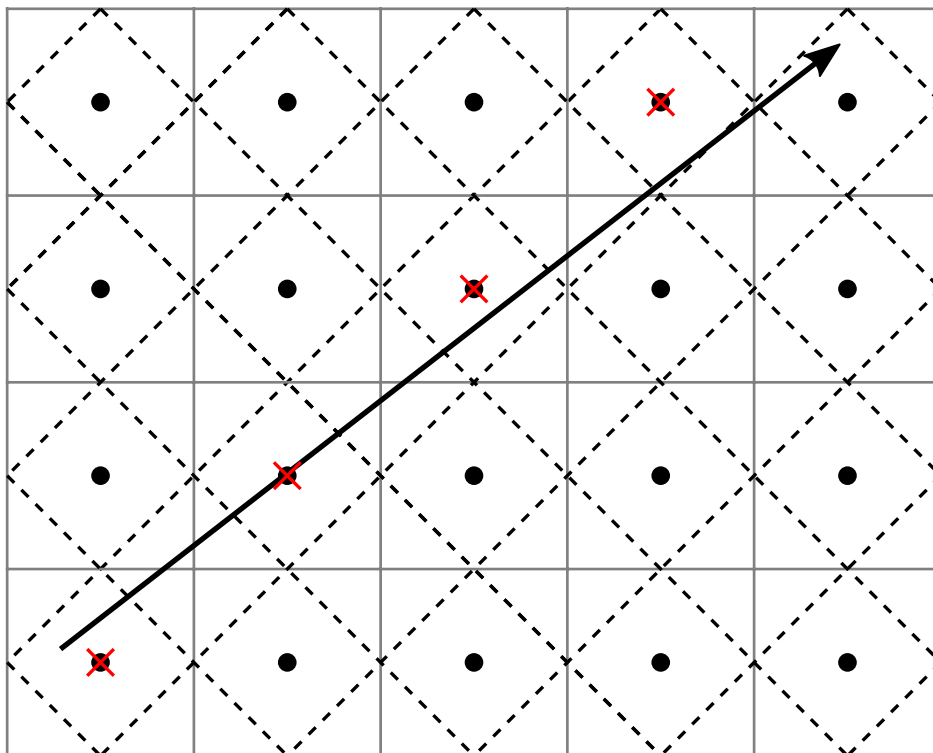


Figure 16. Visualization of Bresenham's algorithm

To avoid difficulties when an endpoint lies on a boundary of R_f we (in principle) perturb the supplied endpoints by a tiny amount. Let p_a and p_b have framebuffer coordinates (x_a, y_a) and (x_b, y_b) , respectively. Obtain the perturbed endpoints p_a' given by $(x_a, y_a) - (\epsilon, \epsilon^2)$ and p_b' given by $(x_b, y_b) - (\epsilon, \epsilon^2)$. Rasterizing the line segment starting at p_a and ending at p_b produces those fragments f for which the segment starting at p_a' and ending on p_b' intersects R_b , except if p_b' is contained in R_f . ϵ is chosen to be so small that rasterizing the line segment produces the same fragments when δ is substituted for ϵ for any $0 < \delta \leq \epsilon$.

When p_a and p_b lie on fragment centers, this characterization of fragments reduces to Bresenham's algorithm with one modification: lines produced in this description are "half-open", meaning that the final fragment (corresponding to p_b) is not drawn. This means that when rasterizing a series of connected line segments, shared endpoints will be produced only once rather than twice (as would occur with Bresenham's algorithm).

Implementations **may** use other line segment rasterization algorithms, subject to the following rules:

- The coordinates of a fragment produced by the algorithm **must** not deviate by more than one unit in either x or y framebuffer coordinates from a corresponding fragment produced by the diamond-exit rule.
- The total number of fragments produced by the algorithm **must** not differ from that produced by the diamond-exit rule by more than one.
- For an x-major line, two fragments that lie in the same framebuffer-coordinate column **must** not be produced (for a y-major line, two fragments that lie in the same framebuffer-coordinate row **must** not be produced).
- If two line segments share a common endpoint, and both segments are either x-major (both left-to-right or both right-to-left) or y-major (both bottom-to-top or both top-to-bottom), then rasterizing both segments **must** not produce duplicate fragments. Fragments also **must** not be omitted so as to interrupt continuity of the connected segments.

The actual width w of Bresenham lines is determined by rounding the line width to the nearest integer, clamping it to the implementation-dependent **LineWidthRange** (with both values rounded to the nearest integer), then clamping it to be no less than 1.

Bresenham line segments of width other than one are rasterized by offsetting them in the minor direction (for an x-major line, the minor direction is y, and for a y-major line, the minor direction is x) and producing a row or column of fragments in the minor direction. If the line segment has endpoints given by (x_0, y_0) and (x_1, y_1) in framebuffer coordinates, the segment with endpoints $(x_0, y_0 - \frac{w-1}{2})$ and $(x_1, y_1 - \frac{w-1}{2})$ is rasterized, but instead of a single fragment, a column of fragments of height w (a row of fragments of length w for a y-major segment) is produced at each x (y for y-major) location. The lowest fragment of this column is the fragment that would be produced by rasterizing the segment of width 1 with the modified coordinates.

The preferred method of attribute interpolation for a wide line is to generate the same attribute values for all fragments in the row or column described above, as if the adjusted line was used for interpolation and those values replicated to the other fragments, except for **FragCoord** which is interpolated as usual. Implementations **may** instead interpolate each fragment according to the formula in [Basic Line Segment Rasterization](#), using the original line segment endpoints.

When Bresenham lines are being rasterized, sample locations **may** all be treated as being at the pixel center (this **may** affect attribute and depth interpolation).

NOTE

The sample locations described above are **not** used for determining coverage, they are only used for things like attribute interpolation. The rasterization rules that determine coverage are defined in terms of whether the line intersects **pixels**, as opposed to the point sampling rules used for other primitive types. So these rules are independent of the sample locations. One consequence of this is that Bresenham lines cover the same pixels regardless of the number of rasterization samples, and cover all samples in those pixels (unless masked out or killed).

26.8.3. Line Stipple

If the `stippledLineEnable` member of `VkPipelineRasterizationLineStateCreateInfo` is `VK_TRUE`, then lines are rasterized with a *line stipple* determined by `lineStippleFactor` and `lineStipplePattern`. `lineStipplePattern` is an unsigned 16-bit integer that determines which fragments are to be drawn or discarded when the line is rasterized. `lineStippleFactor` is a count that is used to modify the effective line stipple by causing each bit in `lineStipplePattern` to be used `lineStippleFactor` times.

Line stippling discards certain fragments that are produced by rasterization. The masking is achieved using three parameters: the 16-bit line stipple pattern p , the line stipple factor r , and an integer stipple counter s . Let

$$b = \lfloor \frac{s}{r} \rfloor \bmod 16$$

Then a fragment is produced if the b 'th bit of p is 1, and discarded otherwise. The bits of p are numbered with 0 being the least significant and 15 being the most significant.

The initial value of s is zero. For `VK_LINE_RASTERIZATION_MODE_BRESENHAM` lines, s is incremented after production of each fragment of a line segment (fragments are produced in order, beginning at the starting point and working towards the ending point). For `VK_LINE_RASTERIZATION_MODE_RECTANGULAR` and `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH` lines, the rectangular region is subdivided into adjacent unit-length rectangles, and s is incremented once for each rectangle. Rectangles with a value of s such that the b 'th bit of p is zero are discarded. If the last rectangle in a line segment is shorter than unit-length, then the remainder **may** carry over to the next line segment in the line strip using the same value of s (this is the preferred behavior, for the stipple pattern to appear more consistent through the strip).

s is reset to 0 at the start of each strip (for line strips), and before every line segment in a group of independent segments.

If the line segment has been clipped, then the value of s at the beginning of the line segment is implementation-dependent.

To [dynamically set](#) the line stipple state, call:

```
// Provided by VK_KHR_line_rasterization
void vkCmdSetLineStippleKHR(
    VkCommandBuffer                      commandBuffer,
    uint32_t                             lineStippleFactor,
    uint16_t                             lineStipplePattern);
```

```
// Provided by VK_EXT_line_rasterization
void vkCmdSetLineStippleEXT(
    VkCommandBuffer                      commandBuffer,
    uint32_t                             lineStippleFactor,
    uint16_t                             lineStipplePattern);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `lineStippleFactor` is the repeat factor used in stippled line rasterization.
- `lineStipplePattern` is the bit pattern used in stippled line rasterization.

This command sets the line stipple state for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_LINE_STIPPLE` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineRasterizationLineStateCreateInfo::lineStippleFactor` and `VkPipelineRasterizationLineStateCreateInfo::lineStipplePattern` values used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetLineStipple-lineStippleFactor-02776
`lineStippleFactor` **must** be in the range [1,256]

Valid Usage (Implicit)

- VUID-vkCmdSetLineStipple-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetLineStipple-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdSetLineStipple-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally

synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHIC S_BIT	State

26.8.4. Smooth Lines

If the `lineRasterizationMode` member of `VkPipelineRasterizationLineStateCreateInfo` is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH`, then lines are considered to be rectangles using the same geometry as for `VK_LINE_RASTERIZATION_MODE_RECTANGULAR` lines. The rules for determining which pixels are covered are implementation-dependent, and **may** include nearby pixels where no sample locations are covered or where the rectangle does not intersect the pixel at all. For each pixel that is considered covered, the fragment computes a coverage value that approximates the area of the intersection of the rectangle with the pixel square, and this coverage value is multiplied into the color location 0's alpha value after fragment shading, as described in [Multisample Coverage](#).

NOTE

The details of the rasterization rules and area calculation are left intentionally vague, to allow implementations to generate coverage and values that are aesthetically pleasing.

26.9. Polygons

A polygon results from the decomposition of a triangle strip, triangle fan or a series of independent triangles. Like points and line segments, polygon rasterization is controlled by several variables in the `VkPipelineRasterizationStateCreateInfo` structure.

26.9.1. Basic Polygon Rasterization

The first step of polygon rasterization is to determine whether the triangle is *back-facing* or *front-facing*. This determination is made based on the sign of the (clipped or unclipped) polygon's area computed in framebuffer coordinates. One way to compute this area is:

$$a = -\frac{1}{2} \sum_{i=0}^{n-1} x_f^i y_f^{i+1} - x_f^{i+1} y_f^i$$

where x_f^i and y_f^i are the x and y framebuffer coordinates of the i th vertex of the n -vertex polygon (vertices are numbered starting at zero for the purposes of this computation) and $i+1$ is $(i+1) \bmod n$.

The interpretation of the sign of a is determined by the `VkPipelineRasterizationStateCreateInfo::frontFace` property of the currently active pipeline. Possible values are:

```
// Provided by VK_VERSION_1_0
typedef enum VkFrontFace {
    VK_FRONT_FACE_COUNTER_CLOCKWISE = 0,
    VK_FRONT_FACE_CLOCKWISE = 1,
} VkFrontFace;
```

- `VK_FRONT_FACE_COUNTER_CLOCKWISE` specifies that a triangle with positive area is considered front-facing.
- `VK_FRONT_FACE_CLOCKWISE` specifies that a triangle with negative area is considered front-facing.

Any triangle which is not front-facing is back-facing, including zero-area triangles.

To [dynamically set](#) the front face orientation, call:

```
// Provided by VK_EXT_extended_dynamic_state
void vkCmdSetFrontFaceEXT(
    VkCommandBuffer          commandBuffer,
    VkFrontFace              frontFace);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `frontFace` is a `VkFrontFace` value specifying the front-facing triangle orientation to be used for culling.

This command sets the front face orientation for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_FRONT_FACE` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineRasterizationStateCreateInfo::frontFace` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetFrontFace-None-08971
At least one of the following **must** be true:
 - the `extendedDynamicState` feature is enabled

Valid Usage (Implicit)

- VUID-vkCmdSetFrontFace-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetFrontFace-frontFace-parameter
`frontFace` **must** be a valid `VkFrontFace` value
- VUID-vkCmdSetFrontFace-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`

- VUID-vkCmdSetFrontFace-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

Once the orientation of triangles is determined, they are culled according to the `VkPipelineRasterizationStateCreateInfo::cullMode` property of the currently active pipeline. Possible values are:

```
// Provided by VK_VERSION_1_0
typedef enum VkCullModeFlagBits {
    VK_CULL_MODE_NONE = 0,
    VK_CULL_MODE_FRONT_BIT = 0x00000001,
    VK_CULL_MODE_BACK_BIT = 0x00000002,
    VK_CULL_MODE_FRONT_AND_BACK = 0x00000003,
} VkCullModeFlagBits;
```

- `VK_CULL_MODE_NONE` specifies that no triangles are discarded
- `VK_CULL_MODE_FRONT_BIT` specifies that front-facing triangles are discarded
- `VK_CULL_MODE_BACK_BIT` specifies that back-facing triangles are discarded
- `VK_CULL_MODE_FRONT_AND_BACK` specifies that all triangles are discarded.

Following culling, fragments are produced for any triangles which have not been discarded.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkCullModeFlags;
```

`VkCullModeFlags` is a bitmask type for setting a mask of zero or more `VkCullModeFlagBits`.

To **dynamically set** the cull mode, call:

```
// Provided by VK_EXT_extended_dynamic_state
void vkCmdSetCullModeEXT(
    VkCommandBuffer                      commandBuffer,
    VkCullModeFlags                     cullMode);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `cullMode` specifies the cull mode property to use for drawing.

This command sets the cull mode for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_CULL_MODE` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineRasterizationStateCreateInfo::cullMode` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetCullMode-None-08971
At least one of the following **must** be true:
 - the `extendedDynamicState` feature is enabled

Valid Usage (Implicit)

- VUID-vkCmdSetCullMode-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetCullMode-cullMode-parameter
`cullMode` **must** be a valid combination of `VkCullModeFlagBits` values
- VUID-vkCmdSetCullMode-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetCullMode-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHIC S_BIT	State

The rule for determining which fragments are produced by polygon rasterization is called *point sampling*. The two-dimensional projection obtained by taking the x and y framebuffer coordinates of the polygon's vertices is formed. Fragments are produced for any fragment area groups of pixels for which any sample points lie inside of this polygon. Coverage bits that correspond to sample points that satisfy the point sampling criteria are 1, other coverage bits are 0. Special treatment is given to a sample whose sample location lies on a polygon edge. In such a case, if two polygons lie on either side of a common edge (with identical endpoints) on which a sample point lies, then exactly one of the polygons **must** result in a covered sample for that fragment during rasterization. As for the data associated with each fragment produced by rasterizing a polygon, we begin by specifying how these values are produced for fragments in a triangle.

Barycentric coordinates are a set of three numbers, a, b, and c, each in the range [0,1], with $a + b + c = 1$. These coordinates uniquely specify any point p within the triangle or on the triangle's boundary as

$$p = a p_a + b p_b + c p_c$$

where p_a , p_b , and p_c are the vertices of the triangle. a, b, and c are determined by:

$$a = \frac{A(p p_b p_c)}{A(p_a p_b p_c)}, \quad b = \frac{A(p p_a p_c)}{A(p_a p_b p_c)}, \quad c = \frac{A(p p_a p_b)}{A(p_a p_b p_c)},$$

where $A(lmn)$ denotes the area in framebuffer coordinates of the triangle with vertices l, m, and n.

Denote an associated datum at p_a , p_b , or p_c as f_a , f_b , or f_c , respectively.

Perspective interpolation for a triangle interpolates three values in a manner that is correct when taking the perspective of the viewport into consideration, by way of the triangle's clip coordinates. An interpolated value f can be determined by

$$f = \frac{a f_a / w_a + b f_b / w_b + c f_c / w_c}{a / w_a + b / w_b + c / w_c}$$

where w_a , w_b , and w_c are the clip w coordinates of p_a , p_b , and p_c , respectively. a, b, and c are the barycentric coordinates of the location at which the data are produced.

Linear interpolation for a triangle directly interpolates three values, and an interpolated value f can be determined by

$$f = a f_a + b f_b + c f_c$$

where f_a , f_b , and f_c are the data associated with p_a , p_b , and p_c , respectively.

The clip coordinate w for a sample is determined using perspective interpolation. The depth value z for a sample is determined using linear interpolation. Interpolation of fragment shader input values are determined by [Interpolation decorations](#).

For a polygon with more than three edges, such as are produced by clipping a triangle, a convex combination of the values of the datum at the polygon's vertices **must** be used to obtain the value assigned to each fragment produced by the rasterization algorithm. That is, it **must** be the case that at every fragment

$$f = \sum_{i=1}^n a_i f_i$$

where n is the number of vertices in the polygon and f_i is the value of f at vertex i . For each i , $0 \leq a_i \leq 1$ and $\sum_{i=1}^n a_i = 1$. The values of a_i **may** differ from fragment to fragment, but at vertex i , $a_i = 1$ and $a_j = 0$ for $j \neq i$.

NOTE

One algorithm that achieves the required behavior is to triangulate a polygon (without adding any vertices) and then treat each triangle individually as already discussed. A scan-line rasterizer that linearly interpolates data along each edge and then linearly interpolates data across each horizontal span from edge to edge also satisfies the restrictions (in this case the numerator and denominator of [perspective interpolation](#) are iterated independently, and a division is performed for each fragment).

26.9.2. Polygon Mode

Possible values of the [VkPipelineRasterizationStateCreateInfo::polygonMode](#) property of the currently active pipeline, specifying the method of rasterization for polygons, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkPolygonMode {
    VK_POLYGON_MODE_FILL = 0,
    VK_POLYGON_MODE_LINE = 1,
    VK_POLYGON_MODE_POINT = 2,
} VkPolygonMode;
```

- [VK_POLYGON_MODE_POINT](#) specifies that polygon vertices are drawn as points.
- [VK_POLYGON_MODE_LINE](#) specifies that polygon edges are drawn as line segments.
- [VK_POLYGON_MODE_FILL](#) specifies that polygons are rendered using the polygon rasterization rules in this section.

These modes affect only the final rasterization of polygons: in particular, a polygon's vertices are shaded and the polygon is clipped and possibly culled before these modes are applied.

The point size of the final rasterization of polygons when `polygon mode` is `VK_POLYGON_MODE_POINT` is implementation-dependent, and the point size **may** either be `PointSize` or 1.0.

26.9.3. Depth Bias

The depth values of all fragments generated by the rasterization of a polygon **can** be biased (offset) by a single depth bias value o that is computed for that polygon.

Depth Bias Enable

The depth bias computation is enabled by the `depthBiasEnable` set with `vkCmdSetDepthBiasEnableEXT` and `vkCmdSetDepthBiasEnableEXT`, or the corresponding `VkPipelineRasterizationStateCreateInfo::depthBiasEnable` value used to create the currently active pipeline. If the depth bias enable is `VK_FALSE`, no bias is applied and the fragment's depth values are unchanged.

To [dynamically enable](#) whether to bias fragment depth values, call:

```
// Provided by VK_EXT_extended_dynamic_state2
void vkCmdSetDepthBiasEnableEXT(
    VkCommandBuffer          commandBuffer,
    VkBool32                 depthBiasEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `depthBiasEnable` controls whether to bias fragment depth values.

This command sets the depth bias enable for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineRasterizationStateCreateInfo::depthBiasEnable` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetDepthBiasEnable-None-08970
At least one of the following **must** be true:
 - the `extendedDynamicState2` feature is enabled

Valid Usage (Implicit)

- VUID-vkCmdSetDepthBiasEnable-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetDepthBiasEnable-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdSetDepthBiasEnable-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

Depth Bias Computation

The depth bias depends on three parameters:

- `depthBiasSlopeFactor` scales the maximum depth slope m of the polygon
- `depthBiasConstantFactor` scales the parameter r of the depth attachment
- the scaled terms are summed to produce a value which is then clamped to a minimum or maximum value specified by `depthBiasClamp`

`depthBiasSlopeFactor`, `depthBiasConstantFactor`, and `depthBiasClamp` **can** each be positive, negative, or zero. These parameters are set as described for `vkCmdSetDepthBias` below.

The maximum depth slope m of a triangle is

$$m = \sqrt{\sum \left| \frac{\partial z_f}{\partial x_f} \right|^2 + \sum \left| \frac{\partial z_f}{\partial y_f} \right|^2}$$

where (x_f, y_f, z_f) is a point on the triangle. m **may** be approximated as

$$m = \max \left\| \frac{\partial z_f}{\partial x_f}, \left| \frac{\partial z_f}{\partial y_f} \right| \right\|.$$

r is the minimum resolvable difference that depends on the depth attachment representation. It is the smallest difference in a sample's depth z_f values that is guaranteed to remain distinct throughout polygon rasterization and in the depth attachment. All pairs of fragments generated by the rasterization of two polygons with otherwise identical vertices, but z_f values that differ by r , will have distinct depth values.

For fixed-point depth attachment representations, r is constant throughout the range of the entire depth attachment. Its value is implementation-dependent but **must** be at most

$$r = 2 \times 2^{-n}$$

where n is the number of bits used for the depth aspect.

For floating-point depth attachment, there is no single minimum resolvable difference. In this case, the minimum resolvable difference for a given polygon is dependent on the maximum exponent, e , in the range of z values spanned by the primitive. If n is the number of bits in the floating-point mantissa, the minimum resolvable difference, r , for the given primitive is defined as

$$r = 2^{e-n}$$

If no depth attachment is present, r is undefined.

The bias value o for a polygon is

$$o = \text{dbclamp}(m \times \text{depthBiasSlopeFactor} + r \times \text{depthBiasConstantFactor})$$

where $\text{dbclamp}(x) = \begin{cases} x & \text{depthBiasClamp} = 0 \text{ or NaN} \\ \min(x, \text{depthBiasClamp}) & \text{depthBiasClamp} > 0 \\ \max(x, \text{depthBiasClamp}) & \text{depthBiasClamp} < 0 \end{cases}$

m is computed as described above. If the depth attachment uses a fixed-point representation, m is a function of depth values in the range $[0,1]$, and o is applied to depth values in the same range.

Depth bias is applied to triangle topology primitives received by the rasterizer regardless of [polygon mode](#). Depth bias **may** also be applied to line and point topology primitives received by the rasterizer.

To [dynamically set](#) the depth bias parameters, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetDepthBias(
    VkCommandBuffer          commandBuffer,
    float                    depthBiasConstantFactor,
    float                    depthBiasClamp,
    float                    depthBiasSlopeFactor);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `depthBiasConstantFactor` is a scalar factor controlling the constant depth value added to each fragment.
- `depthBiasClamp` is the maximum (or minimum) depth bias of a fragment.
- `depthBiasSlopeFactor` is a scalar factor applied to a fragment's slope in depth bias calculations.

This command sets the depth bias parameters for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_BIAS` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the corresponding `VkPipelineRasterizationStateCreateInfo::depthBiasConstantFactor`, `depthBiasClamp`, and `depthBiasSlopeFactor` values used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetDepthBias-depthBiasClamp-00790
If the `depthBiasClamp` feature is not enabled, `depthBiasClamp` **must** be `0.0`

Valid Usage (Implicit)

- VUID-vkCmdSetDepthBias-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetDepthBias-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetDepthBias-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

26.9.4. Conservative Rasterization

If the `pNext` chain of `VkPipelineRasterizationStateCreateInfo` includes a `VkPipelineRasterizationConservativeStateCreateInfoEXT` structure, then that structure includes parameters controlling conservative rasterization.

`VkPipelineRasterizationConservativeStateCreateInfoEXT` is defined as:

```
// Provided by VK_EXT_conservative_rasterization
typedef struct VkPipelineRasterizationConservativeStateCreateInfoEXT {
    VkStructureType                sType;
    const void*                    pNext;
    VkPipelineRasterizationConservativeStateCreateFlagsEXT flags;
    VkConservativeRasterizationModeEXT
conservativeRasterizationMode;
    float
extraPrimitiveOverestimationSize;
} VkPipelineRasterizationConservativeStateCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `conservativeRasterizationMode` is the conservative rasterization mode to use.
- `extraPrimitiveOverestimationSize` is the extra size in pixels to increase the generating primitive during conservative rasterization at each of its edges in *X* and *Y* equally in screen space beyond the base overestimation specified in `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::primitiveOverestimationSize`. If `conservativeRasterizationMode` is not `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT`, this value is ignored.

If this structure is not included in the `pNext` chain, `conservativeRasterizationMode` is considered to be `VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT`, and conservative rasterization is disabled.

Polygon rasterization **can** be made conservative by setting `conservativeRasterizationMode` to `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT` or `VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT` in `VkPipelineRasterizationConservativeStateCreateInfoEXT`.

NOTE

If `conservativePointAndLineRasterization` is supported, conservative rasterization can be applied to line and point primitives, otherwise it must be disabled.

Valid Usage

- VUID-VkPipelineRasterizationConservativeStateCreateInfoEXT-extraPrimitiveOverestimationSize-01769
`extraPrimitiveOverestimationSize` **must** be in the range of `0.0` to `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::maxExtraPrimitiveOverestimationSize` inclusive

Valid Usage (Implicit)

- VUID-VkPipelineRasterizationConservativeStateCreateInfoEXT-sType-sType
`sType` **must** be

VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_CONSERVATIVE_STATE_CREATE_INFO_EXT

- VUID-VkPipelineRasterizationConservativeStateCreateInfoEXT-flags-zero-bitmask
flags must be 0
- VUID-VkPipelineRasterizationConservativeStateCreateInfoEXT-conservativeRasterizationMode-parameter
conservativeRasterizationMode must be a valid [VkConservativeRasterizationModeEXT](#) value

```
// Provided by VK_EXT_conservative_rasterization
typedef VkFlags VkPipelineRasterizationConservativeStateCreateFlagsEXT;
```

VkPipelineRasterizationConservativeStateCreateFlagsEXT is a bitmask type for setting a mask, but is currently reserved for future use.

Possible values of [VkPipelineRasterizationConservativeStateCreateInfoEXT::conservativeRasterizationMode](#), specifying the conservative rasterization mode are:

```
// Provided by VK_EXT_conservative_rasterization
typedef enum VkConservativeRasterizationModeEXT {
    VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT = 0,
    VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT = 1,
    VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT = 2,
} VkConservativeRasterizationModeEXT;
```

- **VK_CONSERVATIVE_RASTERIZATION_MODE_DISABLED_EXT** specifies that conservative rasterization is disabled and rasterization proceeds as normal.
- **VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT** specifies that conservative rasterization is enabled in overestimation mode.
- **VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT** specifies that conservative rasterization is enabled in underestimation mode.

When overestimate conservative rasterization is enabled, rather than evaluating coverage at individual sample locations, a determination is made whether any portion of the pixel (including its edges and corners) is covered by the primitive. If any portion of the pixel is covered, then all bits of the **coverage mask** for the fragment corresponding to that pixel are enabled.

For the purposes of evaluating which pixels are covered by the primitive, implementations **can** increase the size of the primitive by up to **VkPhysicalDeviceConservativeRasterizationPropertiesEXT::primitiveOverestimationSize** pixels at each of the primitive edges. This **may** increase the number of fragments generated by this primitive and represents an overestimation of the pixel coverage.

This overestimation size can be increased further by setting the **extraPrimitiveOverestimationSize** value above **0.0** in steps of **VkPhysicalDeviceConservativeRasterizationPropertiesEXT::extraPrimitiveOverestimationSizeGranularity** up to and including **VkPhysicalDeviceConservativeRasterizationPropertiesEXT::extraPrimitiveOverestimationSize**. This

may further increase the number of fragments generated by this primitive.

The actual precision of the overestimation size used for conservative rasterization **may** vary between implementations and produce results that only approximate the `primitiveOverestimationSize` and `extraPrimitiveOverestimationSizeGranularity` properties.

For triangles if `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT` is enabled, fragments will be generated if the primitive area covers any portion of any pixel inside the fragment area, including their edges or corners. The tie-breaking rule described in [Basic Polygon Rasterization](#) does not apply during conservative rasterization and coverage is set for all fragments generated from shared edges of polygons. Degenerate triangles that evaluate to zero area after rasterization, even for pixels containing a vertex or edge of the zero-area polygon, will be culled if `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::degenerateTrianglesRasterized` is `VK_FALSE` or will generate fragments if `degenerateTrianglesRasterized` is `VK_TRUE`. The fragment input values for these degenerate triangles take their attribute and depth values from the provoking vertex. Degenerate triangles are considered backfacing and the application **can** enable backface culling if desired. Triangles that are zero area before rasterization **may** be culled regardless.

For lines if `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT` is enabled, and the implementation sets `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::conservativePointAndLineRasterization` to `VK_TRUE`, fragments will be generated if the line covers any portion of any pixel inside the fragment area, including their edges or corners. Degenerate lines that evaluate to zero length after rasterization will be culled if `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::degenerateLinesRasterized` is `VK_FALSE` or will generate fragments if `degenerateLinesRasterized` is `VK_TRUE`. The fragments input values for these degenerate lines take their attribute and depth values from the provoking vertex. Lines that are zero length before rasterization **may** be culled regardless.

For points if `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT` is enabled, and the implementation sets `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::conservativePointAndLineRasterization` to `VK_TRUE`, fragments will be generated if the point square covers any portion of any pixel inside the fragment area, including their edges or corners.

When underestimate conservative rasterization is enabled, rather than evaluating coverage at individual sample locations, a determination is made whether all of the pixel (including its edges and corners) is covered by the primitive. If the entire pixel is covered, then a fragment is generated with all bits of its `coverage mask` corresponding to the pixel enabled, otherwise the pixel is not considered covered even if some portion of the pixel is covered. The fragment is discarded if no pixels inside the fragment area are considered covered.

For triangles, if `VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT` is enabled, fragments will only be generated if any pixel inside the fragment area is fully covered by the generating primitive, including its edges and corners.

For lines, if `VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT` is enabled, fragments will be generated if any pixel inside the fragment area, including its edges and corners, are entirely covered by the line.

For points, if `VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT` is enabled, fragments will only

be generated if the point square covers the entirety of any pixel square inside the fragment area, including its edges or corners.

For both overestimate and underestimate conservative rasterization modes a fragment has all of its pixel squares fully covered by the generating primitive **must** set `FullyCoveredEXT` to `VK_TRUE` if the implementation enables the `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::fullyCoveredFragmentShaderInputVariable` feature.

When setting the `fragment shading rate` results in fragments covering multiple pixels, coverage for conservative rasterization is still evaluated on a per-pixel basis and may result in fragments with partial coverage. For fragment shader inputs decorated with `FullyCoveredEXT`, a fragment is considered fully covered if and only if all pixels in the fragment are fully covered by the generating primitive.

Chapter 27. Fragment Operations

Fragments produced by rasterization go through a number of operations to determine whether or how values produced by fragment shading are written to the framebuffer.

The following fragment operations adhere to [rasterization order](#), and are typically performed in this order:

1. [Discard rectangles test](#)
2. [Scissor test](#)
3. [Sample mask test](#)
4. Certain [Fragment shading](#) operations:
 - [Sample Mask Accesses](#)
 - [Depth Replacement](#)
 - [Stencil Reference Replacement](#)
 - [Interlocked Operations](#)
5. [Multisample coverage](#)
6. [Depth bounds test](#)
7. [Stencil test](#)
8. [Depth test](#)
9. [Sample counting](#)
10. [Coverage reduction](#)

The [coverage mask](#) generated by rasterization describes the initial coverage of each sample covered by the fragment. Fragment operations will update the coverage mask to add or subtract coverage where appropriate. If a fragment operation results in all bits of the coverage mask being 0, the fragment is discarded, and no further operations are performed.

When one of the fragment operations in this chapter is described as “replacing” a fragment shader output, that output is replaced unconditionally, even if no fragment shader previously wrote to that output.

If there is a [fragment shader](#) and it declares the [PostDepthCoverage](#) execution mode, the [sample mask test](#) is instead performed after the [depth test](#).

If there is a [fragment shader](#) and it declares the [EarlyFragmentTests](#) execution mode, [fragment shading](#) and [multisample coverage](#) operations **should** instead be performed after [sample counting](#), and [sample mask test](#) **may** instead be performed after [sample counting](#).

For a pipeline with the following properties:

- a fragment shader is specified
- the fragment shader does not write to storage resources;

- the fragment shader specifies the `DepthReplacing` execution mode; and
- either
 - the fragment shader specifies the `DepthUnchanged` execution mode;
 - the fragment shader specifies the `DepthLess` execution mode and the pipeline uses a `VkPipelineDepthStencilStateCreateInfo::depthCompareOp` of `VK_COMPARE_OP_GREATER` or `VK_COMPARE_OP_GREATER_OR_EQUAL`; or
 - the fragment shader specifies the `DepthGreater` execution mode and the pipeline uses a `VkPipelineDepthStencilStateCreateInfo::depthCompareOp` of `VK_COMPARE_OP_LESS` or `VK_COMPARE_OP_LESS_OR_EQUAL`

the implementation **may** perform `depth bounds test` before `fragment shading` and perform an additional `depth test` immediately after that using the interpolated depth value generated by rasterization.

Once all fragment operations have completed, fragment shader outputs for covered color attachment samples pass through `framebuffer operations`.

27.1. Discard Rectangles Test

The discard rectangle test compares the framebuffer coordinates (x_f, y_f) of each sample covered by a fragment against a set of *discard rectangles*.

Each discard rectangle is defined by a `VkRect2D`. These values are either set by the `VkPipelineDiscardRectangleStateCreateInfoEXT` structure during pipeline creation, or dynamically by the `vkCmdSetDiscardRectangleEXT` command.

A given sample is considered inside a discard rectangle if the x_f is in the range [`VkRect2D::offset.x`, `VkRect2D::offset.x + VkRect2D::extent.x`), and y_f is in the range [`VkRect2D::offset.y`, `VkRect2D::offset.y + VkRect2D::extent.y`). If the test is set to be inclusive, samples that are not inside any of the discard rectangles will have their coverage set to 0. If the test is set to be exclusive, samples that are inside any of the discard rectangles will have their coverage set to 0.

If no discard rectangles are specified, the coverage mask is unmodified by this operation.

The `VkPipelineDiscardRectangleStateCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_discard_rectangles
typedef struct VkPipelineDiscardRectangleStateCreateInfoEXT {
    VkStructureType           sType;
    const void*               pNext;
    VkPipelineDiscardRectangleStateCreateFlagsEXT flags;
    VkDiscardRectangleModeEXT discardRectangleMode;
    uint32_t                  discardRectangleCount;
    const VkRect2D*           pDiscardRectangles;
} VkPipelineDiscardRectangleStateCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.

- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `discardRectangleMode` is a `VkDiscardRectangleModeEXT` value determining whether the discard rectangle test is inclusive or exclusive.
- `discardRectangleCount` is the number of discard rectangles to use.
- `pDiscardRectangles` is a pointer to an array of `VkRect2D` structures defining discard rectangles.

If the `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT` dynamic state is enabled for a pipeline, the `pDiscardRectangles` member is ignored. If the `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT` dynamic state is not enabled for the pipeline the presence of this structure in the `VkGraphicsPipelineCreateInfo` chain, and a `discardRectangleCount` greater than zero, implicitly enables discard rectangles in the pipeline, otherwise discard rectangles **must** be enabled or disabled by `vkCmdSetDiscardRectangleEnableEXT`. If the `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT` dynamic state is enabled for the pipeline, the `discardRectangleMode` member is ignored, and the discard rectangle mode **must** be set by `vkCmdSetDiscardRectangleModeEXT`.

When this structure is included in the `pNext` chain of `VkGraphicsPipelineCreateInfo`, it defines parameters of the discard rectangle test. If the `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT` dynamic state is not enabled, and this structure is not included in the `pNext` chain, it is equivalent to specifying this structure with a `discardRectangleCount` of `0`. If all `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT`, `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT`, and `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT` dynamic states are enabled, the application **can** omit this structure from the `pNext` chain of `VkGraphicsPipelineCreateInfo` and still use discard rectangles by setting all state dynamically. In this case `vkCmdSetDiscardRectangleEXT` **must** be called to set the discard rectangle for all indices `[0, maxDiscardRectangles)` before drawing with discard rectangles enabled. Individual discard rectangles **can** be made ineffective by setting their `VkRect2D::extent.width` and `VkRect2D::extent.height` to zero.

Valid Usage

- VUID-VkPipelineDiscardRectangleStateCreateInfoEXT-discardRectangleCount-00582
`discardRectangleCount` **must** be less than or equal to `VkPhysicalDeviceDiscardRectanglePropertiesEXT::maxDiscardRectangles`

Valid Usage (Implicit)

- VUID-VkPipelineDiscardRectangleStateCreateInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_DISCARD_RECTANGLE_STATE_CREATE_INFO_EXT`
- VUID-VkPipelineDiscardRectangleStateCreateInfoEXT-flags-zero bitmask
`flags` **must** be `0`
- VUID-VkPipelineDiscardRectangleStateCreateInfoEXT-discardRectangleMode-parameter
`discardRectangleMode` **must** be a valid `VkDiscardRectangleModeEXT` value

```
// Provided by VK_EXT_discard_rectangles
typedef VkFlags VkPipelineDiscardRectangleStateCreateFlagsEXT;
```

`VkPipelineDiscardRectangleStateCreateFlagsEXT` is a bitmask type for setting a mask, but is currently reserved for future use.

`VkDiscardRectangleModeEXT` values are:

```
// Provided by VK_EXT_discard_rectangles
typedef enum VkDiscardRectangleModeEXT {
    VK_DISCARD_RECTANGLE_MODE_INCLUSIVE_EXT = 0,
    VK_DISCARD_RECTANGLE_MODE_EXCLUSIVE_EXT = 1,
} VkDiscardRectangleModeEXT;
```

- `VK_DISCARD_RECTANGLE_MODE_INCLUSIVE_EXT` specifies that the discard rectangle test is inclusive.
- `VK_DISCARD_RECTANGLE_MODE_EXCLUSIVE_EXT` specifies that the discard rectangle test is exclusive.

To dynamically set the discard rectangles, call:

```
// Provided by VK_EXT_discard_rectangles
void vkCmdSetDiscardRectangleEXT(
    VkCommandBuffer                commandBuffer,
    uint32_t                       firstDiscardRectangle,
    uint32_t                       discardRectangleCount,
    const VkRect2D*                pDiscardRectangles);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `firstDiscardRectangle` is the index of the first discard rectangle whose state is updated by the command.
- `discardRectangleCount` is the number of discard rectangles whose state are updated by the command.
- `pDiscardRectangles` is a pointer to an array of `VkRect2D` structures specifying discard rectangles.

The discard rectangle taken from element `i` of `pDiscardRectangles` replace the current state for the discard rectangle at index `firstDiscardRectangle + i`, for `i` in `[0, discardRectangleCount)`.

This command sets the discard rectangles for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineDiscardRectangleStateCreateInfoEXT::pDiscardRectangles` values used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetDiscardRectangleEXT-firstDiscardRectangle-00585

The sum of `firstDiscardRectangle` and `discardRectangleCount` **must** be less than or equal to `VkPhysicalDeviceDiscardRectanglePropertiesEXT::maxDiscardRectangles`

- VUID-vkCmdSetDiscardRectangleEXT-x-00587
The `x` and `y` member of `offset` in each `VkRect2D` element of `pDiscardRectangles` **must** be greater than or equal to `0`
- VUID-vkCmdSetDiscardRectangleEXT-offset-00588
Evaluation of `(offset.x + extent.width)` in each `VkRect2D` element of `pDiscardRectangles` **must** not cause a signed integer addition overflow
- VUID-vkCmdSetDiscardRectangleEXT-offset-00589
Evaluation of `(offset.y + extent.height)` in each `VkRect2D` element of `pDiscardRectangles` **must** not cause a signed integer addition overflow

Valid Usage (Implicit)

- VUID-vkCmdSetDiscardRectangleEXT-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetDiscardRectangleEXT-pDiscardRectangles-parameter
`pDiscardRectangles` **must** be a valid pointer to an array of `discardRectangleCount` `VkRect2D` structures
- VUID-vkCmdSetDiscardRectangleEXT-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetDiscardRectangleEXT-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdSetDiscardRectangleEXT-discardRectangleCount-arraylength
`discardRectangleCount` **must** be greater than `0`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

To [dynamically set](#) whether discard rectangles are enabled, call:

```
// Provided by VK_EXT_discard_rectangles
void vkCmdSetDiscardRectangleEnableEXT(
    VkCommandBuffer          commandBuffer,
    VkBool32                 discardRectangleEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `discardRectangleEnable` specifies whether discard rectangles are enabled or not.

This command sets the discard rectangle enable for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is implied by the `VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleCount` value used to create the currently active pipeline, where a non-zero `discardRectangleCount` implicitly enables discard rectangles, otherwise they are disabled.

Valid Usage

- VUID-vkCmdSetDiscardRectangleEnableEXT-specVersion-07851
The `VK_EXT_discard_rectangles` extension **must** be enabled, and the implementation **must** support at least `specVersion 2` of this extension

Valid Usage (Implicit)

- VUID-vkCmdSetDiscardRectangleEnableEXT-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetDiscardRectangleEnableEXT-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdSetDiscardRectangleEnableEXT-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

To [dynamically set](#) the discard rectangle mode, call:

```
// Provided by VK_EXT_discard_rectangles
void vkCmdSetDiscardRectangleModeEXT(
    VkCommandBuffer          commandBuffer,
    VkDiscardRectangleModeEXT discardRectangleMode);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `discardRectangleMode` specifies the discard rectangle mode for all discard rectangles, either inclusive or exclusive.

This command sets the discard rectangle mode for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineDiscardRectangleStateCreateInfoEXT::discardRectangleMode` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetDiscardRectangleModeEXT-specVersion-07852
The `VK_EXT_discard_rectangles` extension **must** be enabled, and the implementation **must** support at least `specVersion 2` of this extension

Valid Usage (Implicit)

- VUID-vkCmdSetDiscardRectangleModeEXT-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetDiscardRectangleModeEXT-discardRectangleMode-parameter
`discardRectangleMode` **must** be a valid `VkDiscardRectangleModeEXT` value
- VUID-vkCmdSetDiscardRectangleModeEXT-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdSetDiscardRectangleModeEXT-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

27.2. Scissor Test

The scissor test compares the framebuffer coordinates (x_f, y_f) of each sample covered by a fragment against a *scissor rectangle* at the index equal to the fragment's `ViewportIndex`.

Each scissor rectangle is defined by a `VkRect2D`. These values are either set by the `VkPipelineViewportStateCreateInfo` structure during pipeline creation, or dynamically by the `vkCmdSetScissor` command.

A given sample is considered inside a scissor rectangle if x_f is in the range `[VkRect2D::offset.x, VkRect2D::offset.x + VkRect2D::extent.x)`, and y_f is in the range `[VkRect2D::offset.y, VkRect2D::offset.y + VkRect2D::extent.y)`. Samples with coordinates outside the scissor rectangle at the corresponding `ViewportIndex` will have their coverage set to 0.

To **dynamically set** the scissor rectangles, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetScissor(
    VkCommandBuffer          commandBuffer,
    uint32_t                 firstScissor,
    uint32_t                 scissorCount,
    const VkRect2D*          pScissors);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `firstScissor` is the index of the first scissor whose state is updated by the command.
- `scissorCount` is the number of scissors whose rectangles are updated by the command.
- `pScissors` is a pointer to an array of `VkRect2D` structures defining scissor rectangles.

The scissor rectangles taken from element i of `pScissors` replace the current state for the scissor index `firstScissor + i`, for i in `[0, scissorCount)`.

This command sets the scissor rectangles for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_SCISSOR` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineViewportStateCreateInfo::pScissors` values used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetScissor-firstScissor-00592
The sum of `firstScissor` and `scissorCount` **must** be between `1` and `VkPhysicalDeviceLimits::maxViewports`, inclusive
- VUID-vkCmdSetScissor-firstScissor-00593
If the `multiViewport` feature is not enabled, `firstScissor` **must** be `0`
- VUID-vkCmdSetScissor-scissorCount-00594
If the `multiViewport` feature is not enabled, `scissorCount` **must** be `1`
- VUID-vkCmdSetScissor-x-00595
The `x` and `y` members of `offset` member of any element of `pScissors` **must** be greater than or equal to `0`
- VUID-vkCmdSetScissor-offset-00596
Evaluation of `(offset.x + extent.width)` **must** not cause a signed integer addition overflow for any element of `pScissors`
- VUID-vkCmdSetScissor-offset-00597
Evaluation of `(offset.y + extent.height)` **must** not cause a signed integer addition overflow for any element of `pScissors`

Valid Usage (Implicit)

- VUID-vkCmdSetScissor-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetScissor-pScissors-parameter
`pScissors` **must** be a valid pointer to an array of `scissorCount` `VkRect2D` structures
- VUID-vkCmdSetScissor-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetScissor-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations
- VUID-vkCmdSetScissor-scissorCount-arraylength
`scissorCount` **must** be greater than `0`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized

- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHIC S_BIT	State

27.3. Sample Mask Test

The sample mask test compares the `coverage mask` for a fragment with the *sample mask* defined by `VkPipelineMultisampleStateCreateInfo::pSampleMask`.

Each bit of the coverage mask is associated with a sample index as described in the [rasterization chapter](#). If the bit in `VkPipelineMultisampleStateCreateInfo::pSampleMask` which is associated with that same sample index is set to 0, the coverage mask bit is set to 0.

27.4. Fragment Shading

[Fragment shaders](#) are invoked for each fragment, or as [helper invocations](#).

Most operations in the fragment shader are not performed in [rasterization order](#), with exceptions called out in the following sections.

For fragment shaders invoked by fragments, the following rules apply:

- A fragment shader **must** not be executed if a [fragment operation](#) that executes before fragment shading discards the fragment.
- A fragment shader **may** not be executed if:
 - An implementation determines that another fragment shader, invoked by a subsequent primitive in [primitive order](#), overwrites all results computed by the shader (including writes to storage resources).
 - Any other [fragment operation](#) discards the fragment, and the shader does not write to any storage resources.
 - If a fragment shader statically computes the same values for different framebuffer locations, and does not write to any storage resources, multiple fragments **may** be shaded by one fragment shader invocation. This **may** affect `VK_QUERY_PIPELINE_STATISTIC_FRAGMENT_SHADER_INVOCATIONS_BIT` results, but **must** otherwise not be visible behavior to applications.
- Otherwise, at least one fragment shader **must** be executed.
 - If [sample shading](#) is enabled and multiple invocations per fragment are **required**,

additional invocations **must** be executed as specified.

- Each covered sample **must** be included in at least one fragment shader invocation.

If no fragment shader is included in the pipeline, no fragment shader is executed, and undefined values **may** be written to all color attachment outputs during this fragment operation.

NOTE

Multiple fragment shader invocations may be executed for the same fragment for any number of implementation-dependent reasons. When there is more than one fragment shader invocation per fragment, the association of samples to invocations is implementation-dependent. Stores and atomics performed by these additional invocations have the normal effect.

For example, if the subpass includes multiple views in its view mask, a fragment shader may be invoked separately for each view.

27.4.1. Sample Mask

Reading from the **SampleMask** built-in in the **Input** storage class will return the coverage mask for the current fragment as calculated by fragment operations that executed prior to fragment shading.

If **sample shading** is enabled, fragment shaders will only see values of **1** for samples being shaded - other bits will be **0**.

Each bit of the coverage mask is associated with a sample index as described in the **rasterization chapter**. If the bit in **SampleMask** which is associated with that same sample index is set to **0**, that coverage mask bit is set to **0**.

Values written to the **SampleMask** built-in in the **Output** storage class will be used by the **multisample coverage** operation, with the same encoding as the input built-in.

27.4.2. Shader Termination and Demotion

If a fragment shader is **terminated early** or **demoted to a helper invocation** by using **OpKill**, **OpTerminateInvocation**, or **OpDemoteToHelperInvocation**, samples covered by that shader invocation have their coverage mask bits set to **0**.

27.4.3. Depth Replacement

Writing to the **FragDepth** built-in will replace the fragment's calculated depth values for each sample in the input **SampleMask**. **Depth testing** performed after the fragment shader for this fragment will use this new value as z_f .

27.4.4. Stencil Reference Replacement

Writing to the **FragStencilRefEXT** built-in will replace the fragment's stencil reference value for each sample in the input **SampleMask**. **Stencil testing** performed after the fragment shader for this fragment will use this new value as s_f .

27.4.5. Interlocked Operations

`OpBeginInvocationInterlockEXT` and `OpEndInvocationInterlockEXT` define a section of a fragment shader which imposes additional ordering constraints on operations performed within them. These operations are defined as *interlocked operations*. How interlocked operations are ordered against other fragment shader invocations depends on the specified execution modes.

If the `ShadingRateInterlockOrderedEXT` execution mode is specified, any interlocked operations in a fragment shader **must** happen before interlocked operations in fragment shader invocations that execute later in [rasterization order](#) and cover at least one sample in the same fragment area, and **must** happen after interlocked operations in a fragment shader that executes earlier in [rasterization order](#) and cover at least one sample in the same fragment area.

If the `ShadingRateInterlockUnorderedEXT` execution mode is specified, any interlocked operations in a fragment shader **must** happen before or after interlocked operations in fragment shader invocations that execute earlier or later in [rasterization order](#) and cover at least one sample in the same fragment area.

If the `PixelInterlockOrderedEXT` execution mode is specified, any interlocked operations in a fragment shader **must** happen before interlocked operations in fragment shader invocations that execute later in [rasterization order](#) and cover at least one sample in the same pixel, and **must** happen after interlocked operations in a fragment shader that executes earlier in [rasterization order](#) and cover at least one sample in the same pixel.

If the `PixelInterlockUnorderedEXT` execution mode is specified, any interlocked operations in a fragment shader **must** happen before or after interlocked operations in fragment shader invocations that execute earlier or later in [rasterization order](#) and cover at least one sample in the same pixel.

If the `SampleInterlockOrderedEXT` execution mode is specified, any interlocked operations in a fragment shader **must** happen before interlocked operations in fragment shader invocations that execute later in [rasterization order](#) and cover at least one of the same samples, and **must** happen after interlocked operations in a fragment shader that executes earlier in [rasterization order](#) and cover at least one of the same samples.

If the `SampleInterlockUnorderedEXT` execution mode is specified, any interlocked operations in a fragment shader **must** happen before or after interlocked operations in fragment shader invocations that execute earlier or later in [rasterization order](#) and cover at least one of the same samples.

27.5. Multisample Coverage

If a fragment shader is active and its entry point's interface includes a built-in output variable decorated with `SampleMask`, the coverage mask is **ANDed** with the bits of the `SampleMask` built-in to generate a new coverage mask. If [sample shading](#) is enabled, bits written to `SampleMask` corresponding to samples that are not being shaded by the fragment shader invocation are ignored. If no fragment shader is active, or if the active fragment shader does not include `SampleMask` in its interface, the coverage mask is not modified.

Next, the fragment alpha value and coverage mask are modified based on the line coverage factor if the `lineRasterizationMode` member of the `VkPipelineRasterizationStateCreateInfo` structure is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH`, and the `alphaToCoverageEnable` and `alphaToOneEnable` members of the `VkPipelineMultisampleStateCreateInfo` structure.

All alpha values in this section refer only to the alpha component of the fragment shader output that has a `Location` and `Index` decoration of zero (see the [Fragment Output Interface](#) section). If that shader output has an integer or unsigned integer type, then these operations are skipped.

If the `lineRasterizationMode` member of the `VkPipelineRasterizationStateCreateInfo` structure is `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH` and the fragment came from a line segment, then the alpha value is replaced by multiplying it by the coverage factor for the fragment computed during [smooth line rasterization](#).

If `alphaToCoverageEnable` is enabled, a temporary coverage mask is generated where each bit is determined by the fragment's alpha value, which is ANDed with the fragment coverage mask.

No specific algorithm is specified for converting the alpha value to a temporary coverage mask. It is intended that the number of 1's in this value be proportional to the alpha value (clamped to [0,1]), with all 1's corresponding to a value of 1.0 and all 0's corresponding to 0.0. The algorithm **may** be different at different framebuffer coordinates.

NOTE

Using different algorithms at different framebuffer coordinates **may** help to avoid artifacts caused by regular coverage sample locations.

Finally, if `alphaToOneEnable` is enabled, each alpha value is replaced by the maximum representable alpha value for fixed-point color attachments, or by 1.0 for floating-point attachments. Otherwise, the alpha values are not changed.

27.6. Depth and Stencil Operations

Pipeline state controlling the [depth bounds tests](#), [stencil test](#), and [depth test](#) is specified through the members of the `VkPipelineDepthStencilStateCreateInfo` structure.

The `VkPipelineDepthStencilStateCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineDepthStencilStateCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkPipelineDepthStencilStateCreateFlags flags;
    VkBool32                  depthTestEnable;
    VkBool32                  depthWriteEnable;
    VkCompareOp                depthCompareOp;
    VkBool32                  depthBoundsTestEnable;
    VkBool32                  stencilTestEnable;
    VkStencilOpState           front;
    VkStencilOpState           back;
    float                      minDepthBounds;
    float                      maxDepthBounds;
} VkPipelineDepthStencilStateCreateInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `depthTestEnable` controls whether `depth testing` is enabled.
- `depthWriteEnable` controls whether `depth writes` are enabled when `depthTestEnable` is `VK_TRUE`. Depth writes are always disabled when `depthTestEnable` is `VK_FALSE`.
- `depthCompareOp` is a `VkCompareOp` value specifying the comparison operator to use in the `Depth Comparison` step of the `depth test`.
- `depthBoundsTestEnable` controls whether `depth bounds testing` is enabled.
- `stencilTestEnable` controls whether `stencil testing` is enabled.
- `front` and `back` are `VkStencilOpState` values controlling the corresponding parameters of the `stencil test`.
- `minDepthBounds` is the minimum depth bound used in the `depth bounds test`.
- `maxDepthBounds` is the maximum depth bound used in the `depth bounds test`.

Valid Usage

- VUID-VkPipelineDepthStencilStateCreateInfo-depthBoundsTestEnable-00598
If the `depthBounds` feature is not enabled, `depthBoundsTestEnable` **must** be `VK_FALSE`

Valid Usage (Implicit)

- VUID-VkPipelineDepthStencilStateCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO`
- VUID-VkPipelineDepthStencilStateCreateInfo-pNext-pNext
`pNext` **must** be `NULL`

- `VUID-VkPipelineDepthStencilStateCreateInfo-flags-zerobitmask`
`flags` **must** be `0`
- `VUID-VkPipelineDepthStencilStateCreateInfo-depthCompareOp-parameter`
`depthCompareOp` **must** be a valid `VkCompareOp` value
- `VUID-VkPipelineDepthStencilStateCreateInfo-front-parameter`
`front` **must** be a valid `VkStencilOpState` structure
- `VUID-VkPipelineDepthStencilStateCreateInfo-back-parameter`
`back` **must** be a valid `VkStencilOpState` structure

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineDepthStencilStateCreateFlags;
```

`VkPipelineDepthStencilStateCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

27.7. Depth Bounds Test

The depth bounds test compares the depth value z_a in the depth/stencil attachment at each sample's framebuffer coordinates (x_f, y_f) and `sample index` i against a set of *depth bounds*.

The depth bounds are determined by two floating-point values defining a minimum (`minDepthBounds`) and maximum (`maxDepthBounds`) depth value. These values are either set by the `VkPipelineDepthStencilStateCreateInfo` structure during pipeline creation, or dynamically by `vkCmdSetDepthBoundsTestEnableEXT` and `vkCmdSetDepthBounds`.

A given sample is considered within the depth bounds if z_a is in the range `[minDepthBounds, maxDepthBounds]`. Samples with depth attachment values outside of the depth bounds will have their coverage set to `0`.

If the depth bounds test is disabled, or if there is no depth attachment, the coverage mask is unmodified by this operation.

To `dynamically enable or disable` the depth bounds test, call:

```
// Provided by VK_EXT_extended_dynamic_state
void vkCmdSetDepthBoundsTestEnableEXT(
    VkCommandBuffer          commandBuffer,
    VkBool32                 depthBoundsTestEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `depthBoundsTestEnable` specifies if the depth bounds test is enabled.

This command sets the depth bounds enable for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the

`VkPipelineDepthStencilStateCreateInfo::depthBoundsTestEnable` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetDepthBoundsTestEnable-None-08971
At least one of the following **must** be true:
 - the `extendedDynamicState` feature is enabled
- VUID-vkCmdSetDepthBoundsTestEnable-depthBounds-10010
If the `depthBounds` feature is not enabled, `depthBoundsTestEnable` **must** be `VK_FALSE`

Valid Usage (Implicit)

- VUID-vkCmdSetDepthBoundsTestEnable-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetDepthBoundsTestEnable-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdSetDepthBoundsTestEnable-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

To [dynamically set](#) the depth bounds range, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetDepthBounds(
    VkCommandBuffer          commandBuffer,
    float                    minDepthBounds,
    float                    maxDepthBounds);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `minDepthBounds` is the minimum depth bound.
- `maxDepthBounds` is the maximum depth bound.

This command sets the depth bounds range for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_BOUNDS` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineDepthStencilStateCreateInfo::minDepthBounds` and `VkPipelineDepthStencilStateCreateInfo::maxDepthBounds` values used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetDepthBounds-minDepthBounds-00600
If the `VK_EXT_depth_range_unrestricted` extension is not enabled `minDepthBounds` **must** be between `0.0` and `1.0`, inclusive
- VUID-vkCmdSetDepthBounds-maxDepthBounds-00601
If the `VK_EXT_depth_range_unrestricted` extension is not enabled `maxDepthBounds` **must** be between `0.0` and `1.0`, inclusive
- VUID-vkCmdSetDepthBounds-minDepthBounds-10912
`minDepthBounds` **must** be less than or equal to `maxDepthBounds`

Valid Usage (Implicit)

- VUID-vkCmdSetDepthBounds-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetDepthBounds-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetDepthBounds-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHIC S_BIT	State

27.8. Stencil Test

The stencil test compares the stencil attachment value s_a in the depth/stencil attachment at each sample's framebuffer coordinates (x_f, y_f) and [sample index](#) i against a *stencil reference value*.

If the stencil test is not enabled, as specified by [vkCmdSetStencilTestEnableEXT](#) or [VkPipelineDepthStencilStateCreateInfo::stencilTestEnable](#), or if there is no stencil attachment, the coverage mask is unmodified by this operation.

The stencil test is controlled by one of two sets of stencil-related state, the front stencil state and the back stencil state. Stencil tests and writes use the back stencil state when processing fragments generated by [back-facing polygons](#), and the front stencil state when processing fragments generated by [front-facing polygons](#) or any other primitives.

The comparison operation performed is determined by the [VkCompareOp](#) value set by [vkCmdSetStencilOpEXT::compareOp](#), or by [VkStencilOpState::compareOp](#) during pipeline creation.

The compare mask s_c and stencil reference value s_r of the front or the back stencil state set determine arguments of the comparison operation. s_c is set by the [VkPipelineDepthStencilStateCreateInfo](#) structure during pipeline creation, or by the [vkCmdSetStencilCompareMask](#) command. s_r is set by [VkPipelineDepthStencilStateCreateInfo](#) or by [vkCmdSetStencilReference](#).

s_r and s_a are each independently combined with s_c using a bitwise **AND** operation to create masked reference and attachment values s'_r and s'_a . s'_r and s'_a are used as the *reference* and *test* values, respectively, in the operation specified by the [VkCompareOp](#).

If the comparison evaluates to false, the coverage for the sample is set to **0**.

A new stencil value s_g is generated according to a stencil operation defined by [VkStencilOp](#) parameters set by [vkCmdSetStencilOpEXT](#) or [VkPipelineDepthStencilStateCreateInfo](#). If the stencil test fails, **failOp** defines the stencil operation used. If the stencil test passes however, the stencil op used is based on the [depth test](#) - if it passes, [VkPipelineDepthStencilStateCreateInfo::passOp](#) is used, otherwise [VkPipelineDepthStencilStateCreateInfo::depthFailOp](#) is used.

The stencil attachment value s_a is then updated with the generated stencil value s_g according to the write mask s_w defined by **writeMask** in [VkPipelineDepthStencilStateCreateInfo::front](#) and [VkPipelineDepthStencilStateCreateInfo::back](#) as:

$$s_a = (s_a \oplus \neg s_w) \oplus (s_g \oplus s_w)$$

To **dynamically enable or disable** the stencil test, call:

```
// Provided by VK_EXT_extended_dynamic_state
void vkCmdSetStencilTestEnableEXT(
    VkCommandBuffer          commandBuffer,
    VkBool32                 stencilTestEnable);
```

- **commandBuffer** is the command buffer into which the command will be recorded.
- **stencilTestEnable** specifies if the stencil test is enabled.

This command sets the stencil test enable for subsequent drawing commands when the graphics pipeline is created with **VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE** set in **VkPipelineDynamicStateCreateInfo::pDynamicStates**. Otherwise, this state is specified by the **VkPipelineDepthStencilStateCreateInfo::stencilTestEnable** value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetStencilTestEnable-None-08971
At least one of the following **must** be true:
 - the **extendedDynamicState** feature is enabled

Valid Usage (Implicit)

- VUID-vkCmdSetStencilTestEnable-commandBuffer-parameter
commandBuffer **must** be a valid **VkCommandBuffer** handle
- VUID-vkCmdSetStencilTestEnable-commandBuffer-recording
commandBuffer **must** be in the **recording state**
- VUID-vkCmdSetStencilTestEnable-commandBuffer-cmdpool
The **VkCommandPool** that **commandBuffer** was allocated from **must** support **VK_QUEUE_GRAPHICS_BIT** operations

Host Synchronization

- Host access to **commandBuffer** **must** be externally synchronized
- Host access to the **VkCommandPool** that **commandBuffer** was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

To [dynamically set](#) the stencil operation, call:

```
// Provided by VK_EXT_extended_dynamic_state
void vkCmdSetStencilOpEXT(
    VkCommandBuffer          commandBuffer,
    VkStencilFaceFlags       faceMask,
    VkStencilOp               failOp,
    VkStencilOp               passOp,
    VkStencilOp               depthFailOp,
    VkCompareOp               compareOp);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `faceMask` is a bitmask of `VkStencilFaceFlagBits` specifying the set of stencil state for which to update the stencil operation.
- `failOp` is a `VkStencilOp` value specifying the action performed on samples that fail the stencil test.
- `passOp` is a `VkStencilOp` value specifying the action performed on samples that pass both the depth and stencil tests.
- `depthFailOp` is a `VkStencilOp` value specifying the action performed on samples that pass the stencil test and fail the depth test.
- `compareOp` is a `VkCompareOp` value specifying the comparison operator used in the stencil test.

This command sets the stencil operation for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_STENCIL_OP` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the corresponding `VkPipelineDepthStencilStateCreateInfo::failOp`, `passOp`, `depthFailOp`, and `compareOp` values used to create the currently active pipeline, for both front and back faces.

Valid Usage

- VUID-vkCmdSetStencilOp-None-08971
At least one of the following **must** be true:
 - the `extendedDynamicState` feature is enabled

Valid Usage (Implicit)

- VUID-vkCmdSetStencilOp-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetStencilOp-faceMask-parameter
`faceMask` **must** be a valid combination of `VkStencilFaceFlagBits` values
- VUID-vkCmdSetStencilOp-faceMask-requiredbitmask
`faceMask` **must** not be 0
- VUID-vkCmdSetStencilOp-failOp-parameter
`failOp` **must** be a valid `VkStencilOp` value
- VUID-vkCmdSetStencilOp-passOp-parameter
`passOp` **must** be a valid `VkStencilOp` value
- VUID-vkCmdSetStencilOp-depthFailOp-parameter
`depthFailOp` **must** be a valid `VkStencilOp` value
- VUID-vkCmdSetStencilOp-compareOp-parameter
`compareOp` **must** be a valid `VkCompareOp` value
- VUID-vkCmdSetStencilOp-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdSetStencilOp-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

The `VkStencilOpState` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkStencilOpState {
    VkStencilOp    failOp;
    VkStencilOp    passOp;
    VkStencilOp    depthFailOp;
    VkCompareOp    compareOp;
    uint32_t       compareMask;
    uint32_t       writeMask;
    uint32_t       reference;
} VkStencilOpState;
```

- **failOp** is a [VkStencilOp](#) value specifying the action performed on samples that fail the stencil test.
- **passOp** is a [VkStencilOp](#) value specifying the action performed on samples that pass both the depth and stencil tests.
- **depthFailOp** is a [VkStencilOp](#) value specifying the action performed on samples that pass the stencil test and fail the depth test.
- **compareOp** is a [VkCompareOp](#) value specifying the comparison operator used in the stencil test.
- **compareMask** selects the bits of the unsigned integer stencil values participating in the stencil test.
- **writeMask** selects the bits of the unsigned integer stencil values updated by the stencil test in the stencil framebuffer attachment.
- **reference** is an integer stencil reference value that is used in the unsigned stencil comparison.

Valid Usage (Implicit)

- VUID-VkStencilOpState-failOp-parameter
failOp must be a valid [VkStencilOp](#) value
- VUID-VkStencilOpState-passOp-parameter
passOp must be a valid [VkStencilOp](#) value
- VUID-VkStencilOpState-depthFailOp-parameter
depthFailOp must be a valid [VkStencilOp](#) value
- VUID-VkStencilOpState-compareOp-parameter
compareOp must be a valid [VkCompareOp](#) value

To [dynamically set](#) the stencil compare mask, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetStencilCompareMask(
    VkCommandBuffer          commandBuffer,
    VkStencilFaceFlags        faceMask,
    uint32_t                  compareMask);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `faceMask` is a bitmask of `VkStencilFaceFlagBits` specifying the set of stencil state for which to update the compare mask.
- `compareMask` is the new value to use as the stencil compare mask.

This command sets the stencil compare mask for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkStencilOpState::compareMask` value used to create the currently active pipeline, for both front and back faces.

Valid Usage (Implicit)

- VUID-vkCmdSetStencilCompareMask-commandBuffer-parameter `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetStencilCompareMask-faceMask-parameter `faceMask` **must** be a valid combination of `VkStencilFaceFlagBits` values
- VUID-vkCmdSetStencilCompareMask-faceMask-requiredbitmask `faceMask` **must** not be 0
- VUID-vkCmdSetStencilCompareMask-commandBuffer-recording `commandBuffer` **must** be in the `recording` state
- VUID-vkCmdSetStencilCompareMask-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

`VkStencilFaceFlagBits` values are:

```
// Provided by VK_VERSION_1_0
typedef enum VkStencilFaceFlagBits {
    VK_STENCIL_FACE_FRONT_BIT = 0x00000001,
    VK_STENCIL_FACE_BACK_BIT = 0x00000002,
    VK_STENCIL_FACE_FRONT_AND_BACK = 0x00000003,
} VkStencilFaceFlagBits;
```

- `VK_STENCIL_FACE_FRONT_BIT` specifies that only the front set of stencil state is updated.
- `VK_STENCIL_FACE_BACK_BIT` specifies that only the back set of stencil state is updated.
- `VK_STENCIL_FACE_FRONT_AND_BACK` is the combination of `VK_STENCIL_FACE_FRONT_BIT` and `VK_STENCIL_FACE_BACK_BIT`, and specifies that both sets of stencil state are updated.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkStencilFaceFlags;
```

`VkStencilFaceFlags` is a bitmask type for setting a mask of zero or more `VkStencilFaceFlagBits`.

To [dynamically set](#) the stencil write mask, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetStencilWriteMask(
    VkCommandBuffer          commandBuffer,
    VkStencilFaceFlags       faceMask,
    uint32_t                 writeMask);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `faceMask` is a bitmask of `VkStencilFaceFlagBits` specifying the set of stencil state for which to update the write mask, as described above for `vkCmdSetStencilCompareMask`.
- `writeMask` is the new value to use as the stencil write mask.

This command sets the stencil write mask for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `writeMask` value used to create the currently active pipeline, for both `VkPipelineDepthStencilStateCreateInfo::front` and `VkPipelineDepthStencilStateCreateInfo::back` faces.

Valid Usage (Implicit)

- VUID-vkCmdSetStencilWriteMask-commandBuffer-parameter `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetStencilWriteMask-faceMask-parameter `faceMask` **must** be a valid combination of `VkStencilFaceFlagBits` values

- VUID-vkCmdSetStencilWriteMask-faceMask-requiredbitmask
faceMask must not be 0
- VUID-vkCmdSetStencilWriteMask-commandBuffer-recording
commandBuffer must be in the **recording state**
- VUID-vkCmdSetStencilWriteMask-commandBuffer-cmdpool
The **VkCommandPool** that **commandBuffer** was allocated from **must** support **VK_QUEUE_GRAPHICS_BIT** operations

Host Synchronization

- Host access to **commandBuffer** **must** be externally synchronized
- Host access to the **VkCommandPool** that **commandBuffer** was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

To **dynamically set** the stencil reference value, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetStencilReference(
    VkCommandBuffer          commandBuffer,
    VkStencilFaceFlags       faceMask,
    uint32_t                 reference);
```

- **commandBuffer** is the command buffer into which the command will be recorded.
- **faceMask** is a bitmask of **VkStencilFaceFlagBits** specifying the set of stencil state for which to update the reference value, as described above for **vkCmdSetStencilCompareMask**.
- **reference** is the new value to use as the stencil reference value.

This command sets the stencil reference value for subsequent drawing commands when the graphics pipeline is created with **VK_DYNAMIC_STATE_STENCIL_REFERENCE** set in **VkPipelineDynamicStateCreateInfo::pDynamicStates**. Otherwise, this state is specified by the **VkPipelineDepthStencilStateCreateInfo::reference** value used to create the currently active pipeline, for both front and back faces.

Valid Usage (Implicit)

- VUID-vkCmdSetStencilReference-commandBuffer-parameter **commandBuffer** **must** be a valid [VkCommandBuffer](#) handle
- VUID-vkCmdSetStencilReference-faceMask-parameter **faceMask** **must** be a valid combination of [VkStencilFaceFlagBits](#) values
- VUID-vkCmdSetStencilReference-faceMask-requiredbitmask **faceMask** **must** not be 0
- VUID-vkCmdSetStencilReference-commandBuffer-recording **commandBuffer** **must** be in the [recording state](#)
- VUID-vkCmdSetStencilReference-commandBuffer-cmdpool
The [VkCommandPool](#) that **commandBuffer** was allocated from **must** support [VK_QUEUE_GRAPHICS_BIT](#) operations

Host Synchronization

- Host access to **commandBuffer** **must** be externally synchronized
- Host access to the [VkCommandPool](#) that **commandBuffer** was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

Possible values of the **failOp**, **passOp**, and **depthFailOp** members of [VkStencilOpState](#), specifying what happens to the stored stencil value if this or certain subsequent tests fail or pass, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkStencilOp {
    VK_STENCIL_OP_KEEP = 0,
    VK_STENCIL_OP_ZERO = 1,
    VK_STENCIL_OP_REPLACE = 2,
    VK_STENCIL_OP_INCREMENT_AND_CLAMP = 3,
    VK_STENCIL_OP_DECREMENT_AND_CLAMP = 4,
    VK_STENCIL_OP_INVERT = 5,
    VK_STENCIL_OP_INCREMENT_AND_WRAP = 6,
    VK_STENCIL_OP_DECREMENT_AND_WRAP = 7,
} VkStencilOp;
```


- `VK_STENCIL_OP_KEEP` keeps the current value.
- `VK_STENCIL_OP_ZERO` sets the value to 0.
- `VK_STENCIL_OP_REPLACE` sets the value to *reference*.
- `VK_STENCIL_OP_INCREMENT_AND_CLAMP` increments the current value and clamps to the maximum representable unsigned value.
- `VK_STENCIL_OP_DECREMENT_AND_CLAMP` decrements the current value and clamps to 0.
- `VK_STENCIL_OP_INVERT` bitwise-inverts the current value.
- `VK_STENCIL_OP_INCREMENT_AND_WRAP` increments the current value and wraps to 0 when the maximum value would have been exceeded.
- `VK_STENCIL_OP_DECREMENT_AND_WRAP` decrements the current value and wraps to the maximum possible value when the value would go below 0.

For purposes of increment and decrement, the stencil bits are considered as an unsigned integer.

27.9. Depth Test

The depth test compares the depth value z_a in the depth/stencil attachment at each sample's framebuffer coordinates (x_f, y_f) and [sample index](#) i against the sample's depth value z_f . If there is no depth attachment then the depth test is skipped.

The depth test occurs in three stages, as detailed in the following sections.

27.9.1. Depth Clamping and Range Adjustment

If `VkPipelineRasterizationStateCreateInfo::depthClampEnable` is enabled, z_f is clamped to $[z_{\min}, z_{\max}]$, where $z_{\min} = \min(n, f)$, $z_{\max} = \max(n, f)$, and n and f are the `minDepth` and `maxDepth` depth range values of the viewport used by this fragment, respectively.

Following depth clamping:

- If z_f is not in the range $[z_{\min}, z_{\max}]$, then z_f is undefined following this step.
- If the depth attachment has a fixed-point format and z_f is not in the range $[0, 1]$, then z_f is undefined following this step.

27.9.2. Depth Comparison

If the depth test is not enabled, as specified by `vkCmdSetDepthTestEnableEXT` or `VkPipelineDepthStencilStateCreateInfo::depthTestEnable`, then this step is skipped.

The comparison operation performed is determined by the `VkCompareOp` value set by `vkCmdSetDepthCompareOpEXT`, or by `VkPipelineDepthStencilStateCreateInfo::depthCompareOp` during pipeline creation. z_f and z_a are used as the *reference* and *test* values, respectively, in the operation specified by the `VkCompareOp`.

If the comparison evaluates to false, the coverage for the sample is set to 0.

27.9.3. Depth Attachment Writes

If depth writes are enabled, as specified by `vkCmdSetDepthWriteEnableEXT` or `VkPipelineDepthStencilStateCreateInfo::depthWriteEnable`, and the comparison evaluated to true, the depth attachment value z_a is set to the sample's depth value z_f . If there is no depth attachment, no value is written.

To [dynamically enable or disable](#) the depth test, call:

```
// Provided by VK_EXT_extended_dynamic_state
void vkCmdSetDepthTestEnableEXT(
    VkCommandBuffer          commandBuffer,
    VkBool32                 depthTestEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `depthTestEnable` specifies if the depth test is enabled.

This command sets the depth test enable for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineDepthStencilStateCreateInfo::depthTestEnable` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetDepthTestEnable-None-08971
At least one of the following **must** be true:
 - the `extendedDynamicState` feature is enabled

Valid Usage (Implicit)

- VUID-vkCmdSetDepthTestEnable-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetDepthTestEnable-commandBuffer-recording
`commandBuffer` **must** be in the [recording state](#)
- VUID-vkCmdSetDepthTestEnable-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally

synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHIC S_BIT	State

To [dynamically set](#) the depth compare operator, call:

```
// Provided by VK_EXT_extended_dynamic_state
void vkCmdSetDepthCompareOpEXT(
    VkCommandBuffer          commandBuffer,
    VkCompareOp              depthCompareOp);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `depthCompareOp` is a `VkCompareOp` value specifying the comparison operator used for the [Depth Comparison](#) step of the [depth test](#).

This command sets the depth comparison operator for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineDepthStencilStateCreateInfo::depthCompareOp` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetDepthCompareOp-None-08971
At least one of the following **must** be true:
 - the `extendedDynamicState` feature is enabled

Valid Usage (Implicit)

- VUID-vkCmdSetDepthCompareOp-commandBuffer-parameter `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetDepthCompareOp-depthCompareOp-parameter `depthCompareOp` **must** be a valid `VkCompareOp` value
- VUID-vkCmdSetDepthCompareOp-commandBuffer-recording `commandBuffer` **must** be in the [recording state](#)
- VUID-vkCmdSetDepthCompareOp-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

To [dynamically set](#) the depth write enable, call:

```
// Provided by VK_EXT_extended_dynamic_state
void vkCmdSetDepthWriteEnableEXT(
    VkCommandBuffer          commandBuffer,
    VkBool32                 depthWriteEnable);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `depthWriteEnable` specifies if depth writes are enabled.

This command sets the depth write enable for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineDepthStencilStateCreateInfo::depthWriteEnable` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetDepthWriteEnable-None-08971
At least one of the following **must** be true:
 - the `extendedDynamicState` feature is enabled

Valid Usage (Implicit)

- VUID-vkCmdSetDepthWriteEnable-commandBuffer-parameter

`commandBuffer` **must** be a valid `VkCommandBuffer` handle

- VUID-vkCmdSetDepthWriteEnable-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state

- VUID-vkCmdSetDepthWriteEnable-commandBuffer-cmdpool

The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

27.10. Sample Counting

Occlusion queries use query pool entries to track the number of samples that pass all the per-fragment tests. The mechanism of collecting an occlusion query value is described in [Occlusion Queries](#).

The occlusion query sample counter increments by one for each sample with a coverage value of 1 in each fragment that survives all the per-fragment tests, including scissor, sample mask, alpha to coverage, stencil, and depth tests.

27.11. Coverage Reduction

Coverage reduction takes the coverage information for a fragment and converts that to a boolean coverage value for each color sample in each pixel covered by the fragment.

27.11.1. Pixel Coverage

Coverage for each pixel is first extracted from the total fragment coverage mask. This consists of `rasterizationSamples` unique coverage samples for each pixel in the fragment area, each with a unique `sample index`. If the fragment only contains a single pixel, coverage for the pixel is equivalent to the fragment coverage.

If the `fragment shading rate` is set, and the fragment covers multiple pixels, each pixel's coverage

consists of the coverage samples with a [pixel index](#) matching that pixel, and each sample retains its unique [sample index](#) *i*.

27.11.2. Color Sample Coverage

Once pixel coverage is determined, coverage for each individual color sample corresponding to that pixel is determined.

The number of `rasterizationSamples` is identical to the number of samples in the color attachments. A color sample is covered if the pixel coverage sample with the same [sample index](#) *i* is covered.

Chapter 28. The Framebuffer

28.1. Blending

Blending combines the incoming *source* fragment's R, G, B, and A values with the *destination* R, G, B, and A values of each sample stored in the framebuffer at the fragment's (x_f, y_f) location. If any components are missing in the framebuffer attachment, they are replaced as described in [Component Substitution](#). Blending is performed for each color sample covered by the fragment, rather than just once for each fragment.

Source and destination values are combined according to the [blend operation](#), quadruplets of source and destination weighting factors determined by the [blend factors](#), and a [blend constant](#), to obtain a new set of R, G, B, and A values, as described below.

Blending is computed and applied separately to each color attachment used by the subpass, with separate controls for each attachment.

Prior to performing the blend operation, signed and unsigned normalized fixed-point color components undergo an implied conversion to floating-point as specified by [Conversion from Normalized Fixed-Point to Floating-Point](#). Blending computations are treated as if carried out in floating-point, and basic blend operations are performed with a precision and dynamic range no lower than that used to represent destination components. [Advanced blending operations](#) are performed with a precision and dynamic range no lower than the smaller of that used to represent destination components or that used to represent 16-bit floating-point values.

NOTE

Blending is only defined for floating-point, UNORM, SNORM, and sRGB formats. Within those formats, the implementation may only support blending on some subset of them. Which formats support blending is indicated by `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`.

The pipeline blend state is included in the `VkPipelineColorBlendStateCreateInfo` structure during graphics pipeline creation:

The `VkPipelineColorBlendStateCreateInfo` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineColorBlendStateCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkPipelineColorBlendStateCreateFlags flags;
    VkBool32                  logicOpEnable;
    VkLogicOp                 logicOp;
    uint32_t                  attachmentCount;
    const VkPipelineColorBlendAttachmentState* pAttachments;
    float                     blendConstants[4];
} VkPipelineColorBlendStateCreateInfo;
```


- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.
- `logicOpEnable` controls whether to apply `Logical Operations`.
- `logicOp` selects which logical operation to apply.
- `attachmentCount` is the number of `VkPipelineColorBlendAttachmentState` elements in `pAttachments`.
- `pAttachments` is a pointer to an array of `VkPipelineColorBlendAttachmentState` structures defining blend state for each color attachment.
- `blendConstants` is a pointer to an array of four values used as the R, G, B, and A components of the blend constant that are used in blending, depending on the `blend factor`.

Valid Usage

- VUID-VkPipelineColorBlendStateCreateInfo-pAttachments-00605
If the `independentBlend` feature is not enabled, all elements of `pAttachments` **must** be identical
- VUID-VkPipelineColorBlendStateCreateInfo-logicOpEnable-00606
If the `logicOp` feature is not enabled, `logicOpEnable` **must** be `VK_FALSE`
- VUID-VkPipelineColorBlendStateCreateInfo-logicOpEnable-00607
If `logicOpEnable` is `VK_TRUE`, `logicOp` **must** be a valid `VkLogicOp` value
- VUID-VkPipelineColorBlendStateCreateInfo-pAttachments-07353
If `attachmentCount` is not `0` `pAttachments` **must** be a valid pointer to an array of `attachmentCount` valid `VkPipelineColorBlendAttachmentState` structures

Valid Usage (Implicit)

- VUID-VkPipelineColorBlendStateCreateInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO`
- VUID-VkPipelineColorBlendStateCreateInfo-pNext-pNext
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkPipelineColorBlendAdvancedStateCreateInfoEXT` or `VkPipelineColorWriteCreateInfoEXT`
- VUID-VkPipelineColorBlendStateCreateInfo-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkPipelineColorBlendStateCreateInfo-flags-zeroBitmask
`flags` **must** be `0`
- VUID-VkPipelineColorBlendStateCreateInfo-pAttachments-parameter
If `attachmentCount` is not `0`, and `pAttachments` is not `NULL`, `pAttachments` **must** be a valid pointer to an array of `attachmentCount` valid `VkPipelineColorBlendAttachmentState` structures

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkPipelineColorBlendStateCreateFlags;
```

`VkPipelineColorBlendStateCreateFlags` is a bitmask type for setting a mask, but is currently reserved for future use.

The `VkPipelineColorBlendAttachmentState` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineColorBlendAttachmentState {
    VkBool32          blendEnable;
    VkBlendFactor      srcColorBlendFactor;
    VkBlendFactor      dstColorBlendFactor;
    VkBlendOp          colorBlendOp;
    VkBlendFactor      srcAlphaBlendFactor;
    VkBlendFactor      dstAlphaBlendFactor;
    VkBlendOp          alphaBlendOp;
    VkColorComponentFlags colorWriteMask;
} VkPipelineColorBlendAttachmentState;
```

- `blendEnable` controls whether blending is enabled for the corresponding color attachment. If blending is not enabled, the source fragment's color for that attachment is passed through unmodified.
- `srcColorBlendFactor` selects which blend factor is used to determine the source factors (S_r, S_g, S_b).
- `dstColorBlendFactor` selects which blend factor is used to determine the destination factors (D_r, D_g, D_b).
- `colorBlendOp` selects which blend operation is used to calculate the RGB values to write to the color attachment.
- `srcAlphaBlendFactor` selects which blend factor is used to determine the source factor S_a .
- `dstAlphaBlendFactor` selects which blend factor is used to determine the destination factor D_a .
- `alphaBlendOp` selects which blend operation is used to calculate the alpha values to write to the color attachment.
- `colorWriteMask` is a bitmask of `VkColorComponentFlagBits` specifying which of the R, G, B, and/or A components are enabled for writing, as described for the [Color Write Mask](#).

Valid Usage

- VUID-VkPipelineColorBlendAttachmentState-srcColorBlendFactor-00608
If the `dualSrcBlend` feature is not enabled, `srcColorBlendFactor` **must** not be `VK_BLEND_FACTOR_SRC1_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR`, `VK_BLEND_FACTOR_SRC1_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA`
- VUID-VkPipelineColorBlendAttachmentState-dstColorBlendFactor-00609
If the `dualSrcBlend` feature is not enabled, `dstColorBlendFactor` **must** not be

VK_BLEND_FACTOR_SRC1_COLOR, VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR,
VK_BLEND_FACTOR_SRC1_ALPHA, or VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA

- VUID-VkPipelineColorBlendAttachmentState-srcAlphaBlendFactor-00610
If the `dualSrcBlend` feature is not enabled, `srcAlphaBlendFactor` **must** not be `VK_BLEND_FACTOR_SRC1_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR`, `VK_BLEND_FACTOR_SRC1_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA`
- VUID-VkPipelineColorBlendAttachmentState-dstAlphaBlendFactor-00611
If the `dualSrcBlend` feature is not enabled, `dstAlphaBlendFactor` **must** not be `VK_BLEND_FACTOR_SRC1_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR`, `VK_BLEND_FACTOR_SRC1_ALPHA`, or `VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA`
- VUID-VkPipelineColorBlendAttachmentState-colorBlendOp-01406
If either of `colorBlendOp` or `alphaBlendOp` is an `advanced blend operation`, then `colorBlendOp` **must** equal `alphaBlendOp`
- VUID-VkPipelineColorBlendAttachmentState-advancedBlendIndependentBlend-01407
If `VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT::advancedBlendIndependentBlend` is `VK_FALSE` and `colorBlendOp` is an `advanced blend operation`, then `colorBlendOp` **must** be the same for all attachments
- VUID-VkPipelineColorBlendAttachmentState-advancedBlendIndependentBlend-01408
If `VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT::advancedBlendIndependentBlend` is `VK_FALSE` and `alphaBlendOp` is an `advanced blend operation`, then `alphaBlendOp` **must** be the same for all attachments
- VUID-VkPipelineColorBlendAttachmentState-advancedBlendAllOperations-01409
If `VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT::advancedBlendAllOperations` is `VK_FALSE`, then `colorBlendOp` **must** not be `VK_BLEND_OP_ZERO_EXT`, `VK_BLEND_OP_SRC_EXT`, `VK_BLEND_OP_DST_EXT`, `VK_BLEND_OP_SRC_OVER_EXT`, `VK_BLEND_OP_DST_OVER_EXT`, `VK_BLEND_OP_SRC_IN_EXT`, `VK_BLEND_OP_DST_IN_EXT`, `VK_BLEND_OP_SRC_OUT_EXT`, `VK_BLEND_OP_DST_OUT_EXT`, `VK_BLEND_OP_SRC_ATOP_EXT`, `VK_BLEND_OP_DST_ATOP_EXT`, `VK_BLEND_OP_XOR_EXT`, `VK_BLEND_OP_INVERT_EXT`, `VK_BLEND_OP_INVERT_RGB_EXT`, `VK_BLEND_OP_LINEARODDGE_EXT`, `VK_BLEND_OP_LINEARBURN_EXT`, `VK_BLEND_OP_VIVIDLIGHT_EXT`, `VK_BLEND_OP_LINEARLIGHT_EXT`, `VK_BLEND_OP_PINLIGHT_EXT`, `VK_BLEND_OP_HARDMIX_EXT`, `VK_BLEND_OP_PLUS_EXT`, `VK_BLEND_OP_PLUS_CLAMPED_EXT`, `VK_BLEND_OP_PLUS_CLAMPED_ALPHA_EXT`, `VK_BLEND_OP_PLUS_DARKER_EXT`, `VK_BLEND_OP_MINUS_EXT`, `VK_BLEND_OP_MINUS_CLAMPED_EXT`, `VK_BLEND_OP_CONTRAST_EXT`, `VK_BLEND_OP_INVERT_OVG_EXT`, `VK_BLEND_OP_RED_EXT`, `VK_BLEND_OP_GREEN_EXT`, or `VK_BLEND_OP_BLUE_EXT`
- VUID-VkPipelineColorBlendAttachmentState-colorBlendOp-01410
If `colorBlendOp` or `alphaBlendOp` is an `advanced blend operation`, then `colorAttachmentCount` of the subpass this pipeline is compiled against **must** be less than or equal to `VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT::advancedBlendMaxColorAttachments`

Valid Usage (Implicit)

- VUID-VkPipelineColorBlendAttachmentState-srcColorBlendFactor-parameter
`srcColorBlendFactor` **must** be a valid `VkBlendFactor` value

- VUID-VkPipelineColorBlendAttachmentState-dstColorBlendFactor-parameter **dstColorBlendFactor** must be a valid [VkBlendFactor](#) value
- VUID-VkPipelineColorBlendAttachmentState-colorBlendOp-parameter **colorBlendOp** must be a valid [VkBlendOp](#) value
- VUID-VkPipelineColorBlendAttachmentState-srcAlphaBlendFactor-parameter **srcAlphaBlendFactor** must be a valid [VkBlendFactor](#) value
- VUID-VkPipelineColorBlendAttachmentState-dstAlphaBlendFactor-parameter **dstAlphaBlendFactor** must be a valid [VkBlendFactor](#) value
- VUID-VkPipelineColorBlendAttachmentState-alphaBlendOp-parameter **alphaBlendOp** must be a valid [VkBlendOp](#) value
- VUID-VkPipelineColorBlendAttachmentState-colorWriteMask-parameter **colorWriteMask** must be a valid combination of [VkColorComponentFlagBits](#) values

28.1.1. Blend Factors

The source and destination color and alpha blending factors are selected from the enum:

```
// Provided by VK_VERSION_1_0
typedef enum VkBlendFactor {
    VK_BLEND_FACTOR_ZERO = 0,
    VK_BLEND_FACTOR_ONE = 1,
    VK_BLEND_FACTOR_SRC_COLOR = 2,
    VK_BLEND_FACTOR_ONE_MINUS_SRC_COLOR = 3,
    VK_BLEND_FACTOR_DST_COLOR = 4,
    VK_BLEND_FACTOR_ONE_MINUS_DST_COLOR = 5,
    VK_BLEND_FACTOR_SRC_ALPHA = 6,
    VK_BLEND_FACTOR_ONE_MINUS_SRC_ALPHA = 7,
    VK_BLEND_FACTOR_DST_ALPHA = 8,
    VK_BLEND_FACTOR_ONE_MINUS_DST_ALPHA = 9,
    VK_BLEND_FACTOR_CONSTANT_COLOR = 10,
    VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR = 11,
    VK_BLEND_FACTOR_CONSTANT_ALPHA = 12,
    VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA = 13,
    VK_BLEND_FACTOR_SRC_ALPHA_SATURATE = 14,
    VK_BLEND_FACTOR_SRC1_COLOR = 15,
    VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR = 16,
    VK_BLEND_FACTOR_SRC1_ALPHA = 17,
    VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA = 18,
} VkBlendFactor;
```

The semantics of the enum values are described in the table below:

Table 33. Blend Factors

VkBlendFactor	RGB Blend Factors (S_r, S_g, S_b) or (D_r, D_g, D_b)	Alpha Blend Factor (S_a or D_a)
VK_BLEND_FACTOR_ZERO	(0,0,0)	0
VK_BLEND_FACTOR_ONE	(1,1,1)	1
VK_BLEND_FACTOR_SRC_COLOR	(R_{s0}, G_{s0}, B_{s0})	A_{s0}
VK_BLEND_FACTOR_ONE_MINUS_SRC_COLOR	($1-R_{s0}, 1-G_{s0}, 1-B_{s0}$)	$1-A_{s0}$
VK_BLEND_FACTOR_DST_COLOR	(R_d, G_d, B_d)	A_d
VK_BLEND_FACTOR_ONE_MINUS_DST_COLOR	($1-R_d, 1-G_d, 1-B_d$)	$1-A_d$
VK_BLEND_FACTOR_SRC_ALPHA	(A_{s0}, A_{s0}, A_{s0})	A_{s0}
VK_BLEND_FACTOR_ONE_MINUS_SRC_ALPHA	($1-A_{s0}, 1-A_{s0}, 1-A_{s0}$)	$1-A_{s0}$
VK_BLEND_FACTOR_DST_ALPHA	(A_d, A_d, A_d)	A_d
VK_BLEND_FACTOR_ONE_MINUS_DST_ALPHA	($1-A_d, 1-A_d, 1-A_d$)	$1-A_d$
VK_BLEND_FACTOR_CONSTANT_COLOR	(R_c, G_c, B_c)	A_c
VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR	($1-R_c, 1-G_c, 1-B_c$)	$1-A_c$
VK_BLEND_FACTOR_CONSTANT_ALPHA	(A_c, A_c, A_c)	A_c
VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA	($1-A_c, 1-A_c, 1-A_c$)	$1-A_c$
VK_BLEND_FACTOR_SRC_ALPHA_SATURATE	(f, f, f); $f = \min(A_{s0}, 1-A_d)$	1
VK_BLEND_FACTOR_SRC1_COLOR	(R_{s1}, G_{s1}, B_{s1})	A_{s1}
VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR	($1-R_{s1}, 1-G_{s1}, 1-B_{s1}$)	$1-A_{s1}$
VK_BLEND_FACTOR_SRC1_ALPHA	(A_{s1}, A_{s1}, A_{s1})	A_{s1}
VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA	($1-A_{s1}, 1-A_{s1}, 1-A_{s1}$)	$1-A_{s1}$

In this table, the following conventions are used:

- R_{s0}, G_{s0}, B_{s0} and A_{s0} represent the first source color R, G, B, and A components, respectively, for the fragment output location corresponding to the color attachment being blended.
- R_{s1}, G_{s1}, B_{s1} and A_{s1} represent the second source color R, G, B, and A components, respectively, used in dual source blending modes, for the fragment output location corresponding to the color attachment being blended.
- R_d, G_d, B_d and A_d represent the R, G, B, and A components of the destination color. That is, the color currently in the corresponding color attachment for this fragment/sample.
- R_c, G_c, B_c and A_c represent the blend constant R, G, B, and A components, respectively.

To [dynamically set and change](#) the blend constants, call:

```
// Provided by VK_VERSION_1_0
void vkCmdSetBlendConstants(
    VkCommandBuffer                      commandBuffer,
    const float                          blendConstants[4]);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `blendConstants` is a pointer to an array of four values specifying the R_c , G_c , B_c , and A_c components of the blend constant color used in blending, depending on the `blend factor`.

This command sets blend constants for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_BLEND_CONSTANTS` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineColorBlendStateCreateInfo::blendConstants` values used to create the currently active pipeline.

Valid Usage (Implicit)

- VUID-vkCmdSetBlendConstants-commandBuffer-parameter `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetBlendConstants-commandBuffer-recording `commandBuffer` **must** be in the `recording state`
- VUID-vkCmdSetBlendConstants-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

28.1.2. Dual-Source Blending

Blend factors that use the secondary color input ($R_{s1}, G_{s1}, B_{s1}, A_{s1}$) (`VK_BLEND_FACTOR_SRC1_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR`, `VK_BLEND_FACTOR_SRC1_ALPHA`, and `VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA`) **may** consume implementation resources that could

otherwise be used for rendering to multiple color attachments. Therefore, the number of color attachments that **can** be used in a framebuffer **may** be lower when using dual-source blending.

Dual-source blending is only supported if the `dualSrcBlend` feature is enabled.

The maximum number of color attachments that **can** be used in a subpass when using dual-source blending functions is implementation-dependent and is reported as the `maxFragmentDualSrcAttachments` member of `VkPhysicalDeviceLimits`.

Color outputs **can** be bound to the first and second inputs of the blender using the `Index` decoration, as described in [Fragment Output Interface](#). If the second color input to the blender is not written in the shader, or if no output is bound to the second input of a blender, the value of the second input is undefined.

28.1.3. Blend Operations

Once the source and destination blend factors have been selected, they along with the source and destination components are passed to the blending operations. RGB and alpha components **can** use different operations. Possible values of `VkBlendOp`, specifying the operations, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkBlendOp {
    VK_BLEND_OP_ADD = 0,
    VK_BLEND_OP_SUBTRACT = 1,
    VK_BLEND_OP_REVERSE_SUBTRACT = 2,
    VK_BLEND_OP_MIN = 3,
    VK_BLEND_OP_MAX = 4,
    // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_ZERO_EXT = 1000148000,
    // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_SRC_EXT = 1000148001,
    // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_DST_EXT = 1000148002,
    // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_SRC_OVER_EXT = 1000148003,
    // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_DST_OVER_EXT = 1000148004,
    // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_SRC_IN_EXT = 1000148005,
    // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_DST_IN_EXT = 1000148006,
    // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_SRC_OUT_EXT = 1000148007,
    // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_DST_OUT_EXT = 1000148008,
    // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_SRC_ATOP_EXT = 1000148009,
    // Provided by VK_EXT_blend_operation_advanced
    VK_BLEND_OP_DST_ATOP_EXT = 1000148010,
    // Provided by VK_EXT_blend_operation_advanced
```



```
VK_BLEND_OP_XOR_EXT = 1000148011,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_MULTIPLY_EXT = 1000148012,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_SCREEN_EXT = 1000148013,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_OVERLAY_EXT = 1000148014,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_DARKEN_EXT = 1000148015,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_LIGHTEN_EXT = 1000148016,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_COLOR Dodge_EXT = 1000148017,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_COLOR BURN_EXT = 1000148018,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_HARDLIGHT_EXT = 1000148019,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_SOFTLIGHT_EXT = 1000148020,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_DIFFERENCE_EXT = 1000148021,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_EXCLUSION_EXT = 1000148022,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_INVERT_EXT = 1000148023,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_INVERT_RGB_EXT = 1000148024,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_LINEAR Dodge_EXT = 1000148025,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_LINEAR BURN_EXT = 1000148026,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_VIVIDLIGHT_EXT = 1000148027,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_LINEARLIGHT_EXT = 1000148028,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_PINLIGHT_EXT = 1000148029,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_HARDMIX_EXT = 1000148030,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_HSL_HUE_EXT = 1000148031,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_HSL_SATURATION_EXT = 1000148032,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_HSL_COLOR_EXT = 1000148033,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_HSL_LUMINOSITY_EXT = 1000148034,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_PLUS_EXT = 1000148035,  
// Provided by VK_EXT_blend_operation_advanced  
VK_BLEND_OP_PLUS_CLAMPED_EXT = 1000148036,
```

```
// Provided by VK_EXT_blend_operation_advanced
VK_BLEND_OP_PLUS_CLAMPED_ALPHA_EXT = 1000148037,
// Provided by VK_EXT_blend_operation_advanced
VK_BLEND_OP_PLUS_DARKER_EXT = 1000148038,
// Provided by VK_EXT_blend_operation_advanced
VK_BLEND_OP_MINUS_EXT = 1000148039,
// Provided by VK_EXT_blend_operation_advanced
VK_BLEND_OP_MINUS_CLAMPED_EXT = 1000148040,
// Provided by VK_EXT_blend_operation_advanced
VK_BLEND_OP_CONTRAST_EXT = 1000148041,
// Provided by VK_EXT_blend_operation_advanced
VK_BLEND_OP_INVERT_OVG_EXT = 1000148042,
// Provided by VK_EXT_blend_operation_advanced
VK_BLEND_OP_RED_EXT = 1000148043,
// Provided by VK_EXT_blend_operation_advanced
VK_BLEND_OP_GREEN_EXT = 1000148044,
// Provided by VK_EXT_blend_operation_advanced
VK_BLEND_OP_BLUE_EXT = 1000148045,
} VkBlendOp;
```

The semantics of the basic blend operations are described in the table below:

Table 34. Basic Blend Operations

VkBlendOp	RGB Components	Alpha Component
VK_BLEND_OP_ADD	$R = R_{s0} \times S_r + R_d \times D_r$ $G = G_{s0} \times S_g + G_d \times D_g$ $B = B_{s0} \times S_b + B_d \times D_b$	$A = A_{s0} \times S_a + A_d \times D_a$
VK_BLEND_OP_SUBTRACT	$R = R_{s0} \times S_r - R_d \times D_r$ $G = G_{s0} \times S_g - G_d \times D_g$ $B = B_{s0} \times S_b - B_d \times D_b$	$A = A_{s0} \times S_a - A_d \times D_a$
VK_BLEND_OP_REVERSE_SUBTRACT	$R = R_d \times D_r - R_{s0} \times S_r$ $G = G_d \times D_g - G_{s0} \times S_g$ $B = B_d \times D_b - B_{s0} \times S_b$	$A = A_d \times D_a - A_{s0} \times S_a$
VK_BLEND_OP_MIN	$R = \min(R_{s0}, R_d)$ $G = \min(G_{s0}, G_d)$ $B = \min(B_{s0}, B_d)$	$A = \min(A_{s0}, A_d)$
VK_BLEND_OP_MAX	$R = \max(R_{s0}, R_d)$ $G = \max(G_{s0}, G_d)$ $B = \max(B_{s0}, B_d)$	$A = \max(A_{s0}, A_d)$

In this table, the following conventions are used:

- R_{s0} , G_{s0} , B_{s0} and A_{s0} represent the first source color R, G, B, and A components, respectively.
- R_d , G_d , B_d and A_d represent the R, G, B, and A components of the destination color. That is, the color currently in the corresponding color attachment for this fragment/sample.
- S_r , S_g , S_b and S_a represent the source blend factor R, G, B, and A components, respectively.
- D_r , D_g , D_b and D_a represent the destination blend factor R, G, B, and A components, respectively.

The blending operation produces a new set of values R, G, B and A, which are written to the framebuffer attachment. If blending is not enabled for this attachment, then R, G, B and A are assigned R_{s0} , G_{s0} , B_{s0} and A_{s0} , respectively.

If the color attachment is fixed-point, the components of the source and destination values and blend factors are each clamped to [0,1] or [-1,1] respectively for an unsigned normalized or signed normalized color attachment prior to evaluating the blend operations. If the color attachment is floating-point, no clamping occurs.

If the [numeric format](#) of a framebuffer attachment uses sRGB encoding, the R, G, and B destination color values (after conversion from fixed-point to floating-point) are considered to be encoded for the sRGB color space and hence are linearized prior to their use in blending. Each R, G, and B component is converted from nonlinear to linear as described in the “sRGB EOTF” section of the [Khronos Data Format Specification](#). If the format is not sRGB, no linearization is performed.

If the [numeric format](#) of a framebuffer attachment uses sRGB encoding, then the final R, G and B values are converted into the nonlinear sRGB representation before being written to the framebuffer attachment as described in the “sRGB EOTF⁻¹” section of the Khronos Data Format

Specification.

If the [numeric format](#) of a framebuffer color attachment is not sRGB encoded then the resulting c_s values for R, G and B are unmodified. The value of A is never sRGB encoded. That is, the alpha component is always stored in memory as linear.

If the framebuffer color attachment is `VK_ATTACHMENT_UNUSED`, no writes are performed through that attachment. Writes are not performed to framebuffer color attachments greater than or equal to the `VkSubpassDescription::colorAttachmentCount` or `VkSubpassDescription2::colorAttachmentCount` value.

28.1.4. Advanced Blend Operations

The *advanced blend operations* are those listed in tables [f/X/Y/Z Advanced Blend Operations](#), [Hue-Saturation-Luminosity Advanced Blend Operations](#), and [Additional RGB Blend Operations](#).

If the `pNext` chain of `VkPipelineColorBlendStateCreateInfo` includes a `VkPipelineColorBlendAdvancedStateCreateInfoEXT` structure, then that structure includes parameters that affect advanced blend operations.

The `VkPipelineColorBlendAdvancedStateCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_blend_operation_advanced
typedef struct VkPipelineColorBlendAdvancedStateCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkBool32           srcPremultiplied;
    VkBool32           dstPremultiplied;
    VkBlendOverlapEXT  blendOverlap;
} VkPipelineColorBlendAdvancedStateCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `srcPremultiplied` specifies whether the source color of the blend operation is treated as premultiplied.
- `dstPremultiplied` specifies whether the destination color of the blend operation is treated as premultiplied.
- `blendOverlap` is a `VkBlendOverlapEXT` value specifying how the source and destination sample's coverage is correlated.

If this structure is not present, `srcPremultiplied` and `dstPremultiplied` are both considered to be `VK_TRUE`, and `blendOverlap` is considered to be `VK_BLEND_OVERLAP_UNCORRELATED_EXT`.

Valid Usage

- VUID-VkPipelineColorBlendAdvancedStateCreateInfoEXT-srcPremultiplied-01424
If the [non-premultiplied source color](#) property is not supported, `srcPremultiplied` **must** be

VK_TRUE

- VUID-VkPipelineColorBlendAdvancedStateCreateInfoEXT-dstPremultiplied-01425
If the **non-premultiplied destination color** property is not supported, **dstPremultiplied** must be VK_TRUE
- VUID-VkPipelineColorBlendAdvancedStateCreateInfoEXT-blendOverlap-01426
If the **correlated overlap** property is not supported, **blendOverlap** must be VK_BLEND_OVERLAP_UNCORRELATED_EXT

Valid Usage (Implicit)

- VUID-VkPipelineColorBlendAdvancedStateCreateInfoEXT-sType-sType
sType must be VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_ADVANCED_STATE_CREATE_INFO_EXT
- VUID-VkPipelineColorBlendAdvancedStateCreateInfoEXT-blendOverlap-parameter
blendOverlap must be a valid **VkBlendOverlapEXT** value

When using one of the operations in table [f/X/Y/Z Advanced Blend Operations](#) or [Hue-Saturation-Luminosity Advanced Blend Operations](#), blending is performed according to the following equations:

$$\begin{aligned} R &= f(R'_s, R'_d) * p_0(A_s, A_d) + Y * R'_s * p_1(A_s, A_d) + Z * R'_d * p_2(A_s, A_d) \\ G &= f(G'_s, G'_d) * p_0(A_s, A_d) + Y * G'_s * p_1(A_s, A_d) + Z * G'_d * p_2(A_s, A_d) \\ B &= f(B'_s, B'_d) * p_0(A_s, A_d) + Y * B'_s * p_1(A_s, A_d) + Z * B'_d * p_2(A_s, A_d) \\ A &= X * p_0(A_s, A_d) + Y * p_1(A_s, A_d) + Z * p_2(A_s, A_d) \end{aligned}$$

where the function *f* and terms *X*, *Y*, and *Z* are specified in the table. The *R*, *G*, and *B* components of the source color used for blending are derived according to **srcPremultiplied**. If **srcPremultiplied** is VK_TRUE, the fragment color components are considered to have been premultiplied by the *A* component prior to blending. The base source color (*R_s'*, *G_s'*, *B_s'*) is obtained by dividing through by the *A* component:

$$(R'_s, G'_s, B'_s) = \begin{cases} (0, 0, 0) & A_s = 0 \\ \left(\frac{R_s}{A_s}, \frac{G_s}{A_s}, \frac{B_s}{A_s}\right) & \text{otherwise} \end{cases}$$

If **srcPremultiplied** is VK_FALSE, the fragment color components are used as the base color:

$$(R'_s, G'_s, B'_s) = (R_s, G_s, B_s)$$

The *R*, *G*, and *B* components of the destination color used for blending are derived according to **dstPremultiplied**. If **dstPremultiplied** is VK_TRUE, the destination components are considered to have been premultiplied by the *A* component prior to blending. The base destination color (*R_d'*, *G_d'*, *B_d'*) is obtained by dividing through by the *A* component:

$$\{R_d', G_d', B_d'\} = \begin{cases} \{0, 0, 0\} & A_d = 0 \\ \{\frac{R_d}{A_d}, \frac{G_d}{A_d}, \frac{B_d}{A_d}\} & \text{otherwise} \end{cases}$$

If `dstPremultiplied` is `VK_FALSE`, the destination color components are used as the base color:

$$\{R_d', G_d', B_d'\} = \{R_d, G_d, B_d\}$$

When blending using advanced blend operations, we expect that the R, G, and B components of premultiplied source and destination color inputs be stored as the product of non-premultiplied R, G, and B component values and the A component of the color. If any R, G, or B component of a premultiplied input color is non-zero and the A component is zero, the color is considered ill-formed, and the corresponding component of the blend result is undefined.

All of the advanced blend operation formulas in this chapter compute the result as a premultiplied color. If `dstPremultiplied` is `VK_FALSE`, that result color's R, G, and B components are divided by the A component before being written to the framebuffer. If any R, G, or B component of the color is non-zero and the A component is zero, the result is considered ill-formed, and the corresponding component of the blend result is undefined. If all components are zero, that value is unchanged.

If the A component of any input or result color is less than zero, the color is considered ill-formed, and all components of the blend result are undefined.

The weighting functions p_0 , p_1 , and p_2 are defined in table [Advanced Blend Overlap Modes](#). In these functions, the A components of the source and destination colors are taken to indicate the portion of the pixel covered by the fragment (source) and the fragments previously accumulated in the pixel (destination). The functions p_0 , p_1 , and p_2 approximate the relative portion of the pixel covered by the intersection of the source and destination, covered only by the source, and covered only by the destination, respectively.

Possible values of `VkPipelineColorBlendAdvancedStateCreateInfoEXT::blendOverlap`, specifying the blend overlap functions, are:

```
// Provided by VK_EXT_blend_operation_advanced
typedef enum VkBlendOverlapEXT {
    VK_BLEND_OVERLAP_UNCORRELATED_EXT = 0,
    VK_BLEND_OVERLAP_DISJOINT_EXT = 1,
    VK_BLEND_OVERLAP_CONJOINT_EXT = 2,
} VkBlendOverlapEXT;
```

- `VK_BLEND_OVERLAP_UNCORRELATED_EXT` specifies that there is no correlation between the source and destination coverage.
- `VK_BLEND_OVERLAP_CONJOINT_EXT` specifies that the source and destination coverage are considered to have maximal overlap.
- `VK_BLEND_OVERLAP_DISJOINT_EXT` specifies that the source and destination coverage are considered to have minimal overlap.

Table 35. Advanced Blend Overlap Modes

Overlap Mode	Weighting Equations
VK_BLEND_OVERLAP_UNCORRELATED_EXT	$p_0(A_s, A_d) = A_s A_d$ $p_1(A_s, A_d) = A_s(1 - A_d)$ $p_2(A_s, A_d) = A_d(1 - A_s)$
VK_BLEND_OVERLAP_CONJOINT_EXT	$p_0(A_s, A_d) = \min(A_s, A_d)$ $p_1(A_s, A_d) = \max(A_s - A_d, 0)$ $p_2(A_s, A_d) = \max(A_d - A_s, 0)$
VK_BLEND_OVERLAP_DISJOINT_EXT	$p_0(A_s, A_d) = \max(A_s + A_d - 1, 0)$ $p_1(A_s, A_d) = \min(A_s, 1 - A_d)$ $p_2(A_s, A_d) = \min(A_d, 1 - A_s)$

Table 36. f/X/Y/Z Advanced Blend Operations

Mode	Blend Coefficients
VK_BLEND_OP_ZERO_EXT	$(X, Y, Z) = (0, 0, 0)$ $f(C_s, C_d) = 0$
VK_BLEND_OP_SRC_EXT	$(X, Y, Z) = (1, 1, 0)$ $f(C_s, C_d) = C_s$
VK_BLEND_OP_DST_EXT	$(X, Y, Z) = (1, 0, 1)$ $f(C_s, C_d) = C_d$
VK_BLEND_OP_SRC_OVER_EXT	$(X, Y, Z) = (1, 1, 1)$ $f(C_s, C_d) = C_s$
VK_BLEND_OP_DST_OVER_EXT	$(X, Y, Z) = (1, 1, 1)$ $f(C_s, C_d) = C_d$
VK_BLEND_OP_SRC_IN_EXT	$(X, Y, Z) = (1, 0, 0)$ $f(C_s, C_d) = C_s$
VK_BLEND_OP_DST_IN_EXT	$(X, Y, Z) = (1, 0, 0)$ $f(C_s, C_d) = C_d$
VK_BLEND_OP_SRC_OUT_EXT	$(X, Y, Z) = (0, 1, 0)$ $f(C_s, C_d) = 0$
VK_BLEND_OP_DST_OUT_EXT	$(X, Y, Z) = (0, 0, 1)$ $f(C_s, C_d) = 0$
VK_BLEND_OP_SRC_ATOP_EXT	$(X, Y, Z) = (1, 0, 1)$ $f(C_s, C_d) = C_s$
VK_BLEND_OP_DST_ATOP_EXT	$(X, Y, Z) = (1, 1, 0)$ $f(C_s, C_d) = C_d$

Mode	Blend Coefficients
VK_BLEND_OP_XOR_EXT	$\vec{X}, Y, Z = \vec{0}, 1, 1$ $f(C_s, C_d) = 0$
VK_BLEND_OP_MULTIPLY_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = C_s C_d$
VK_BLEND_OP_SCREEN_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = C_s + C_d - C_s C_d$
VK_BLEND_OP_OVERLAY_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \begin{cases} 2C_s C_d & C_d \leq 0.5 \\ 1 - 2(1 - C_s)(1 - C_d) & \text{otherwise} \end{cases}$
VK_BLEND_OP_DARKEN_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \min(C_s, C_d)$
VK_BLEND_OP_LIGHTEN_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \max(C_s, C_d)$
VK_BLEND_OP_COLOR Dodge_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \begin{cases} 0 & C_d \leq 0 \\ \min(1, \frac{C_d}{1 - C_s}) & C_d > 0 \text{ and } C_s < 1 \\ 1 & C_d > 0 \text{ and } C_s \geq 1 \end{cases}$
VK_BLEND_OP_COLOR BURN_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \begin{cases} 1 & C_d \geq 1 \\ 1 - \min(1, \frac{1 - C_d}{C_s}) & C_d < 1 \text{ and } C_s > 0 \\ 0 & C_d < 1 \text{ and } C_s \leq 0 \end{cases}$
VK_BLEND_OP_HARDLIGHT_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \begin{cases} 2C_s C_d & C_s \leq 0.5 \\ 1 - 2(1 - C_s)(1 - C_d) & \text{otherwise} \end{cases}$
VK_BLEND_OP_SOFTLIGHT_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \begin{cases} C_d - (1 - 2C_s)(C_d - C_d^2) & C_s \leq 0.5 \\ C_d + (2C_s - 1)(C_d(1 - 6C_d + 3)) & C_s > 0.5 \text{ and } C_d \leq 0.25 \\ C_d + (2C_s - 1)(C_d - C_d^2) & C_s > 0.5 \text{ and } C_d > 0.25 \end{cases}$
VK_BLEND_OP_DIFFERENCE_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = C_d - C_s $
VK_BLEND_OP_EXCLUSION_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = C_s + C_d - 2C_s C_d$
VK_BLEND_OP_INVERT_EXT	$\vec{X}, Y, Z = \vec{1}, 0, 1$ $f(C_s, C_d) = 1 - C_d$
VK_BLEND_OP_INVERT_RGB_EXT	$\vec{X}, Y, Z = \vec{1}, 0, 1$ $f(C_s, C_d) = C_s(1 - C_d)$
VK_BLEND_OP_LINEAR Dodge_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \begin{cases} C_s + C_d & C_s + C_d \leq 1 \\ 1 & \text{otherwise} \end{cases}$

Mode	Blend Coefficients
VK_BLEND_OP_LINEARBURN_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \begin{cases} C_s + C_d - 1 & C_s + C_d > 1 \\ 0 & \text{otherwise} \end{cases}$
VK_BLEND_OP_VIVIDLIGHT_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \begin{cases} 1 - \min(1, \frac{1 - C_d}{2C_s}) & 0 < C_s < 0.5 \\ 0 & C_s \leq 0 \\ \min(1, \frac{C_d}{2(1 - C_s)}) & 0.5 \leq C_s < 1 \\ 1 & C_s \geq 1 \end{cases}$
VK_BLEND_OP_LINEARLIGHT_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \begin{cases} 1 & 2C_s + C_d > 2 \\ 2C_s + C_d - 1 & 1 < 2C_s + C_d \leq 2 \\ 0 & 2C_s + C_d \leq 1 \end{cases}$
VK_BLEND_OP_PINLIGHT_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \begin{cases} 0 & 2C_s - 1 > C_d \text{ and } C_s < 0.5 \\ 2C_s - 1 & 2C_s - 1 > C_d \text{ and } C_s \geq 0.5 \\ 2C_s & 2C_s - 1 \leq C_d \text{ and } C_s < 0.5C_d \\ C_d & 2C_s - 1 \leq C_d \text{ and } C_s \geq 0.5C_d \end{cases}$
VK_BLEND_OP_HARDMIX_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f(C_s, C_d) = \begin{cases} 0 & C_s + C_d < 1 \\ 1 & \text{otherwise} \end{cases}$

When using one of the HSL blend operations in table [Hue-Saturation-Luminosity Advanced Blend Operations](#) as the blend operation, the RGB color components produced by the function f are effectively obtained by converting both the non-premultiplied source and destination colors to the HSL (hue, saturation, luminosity) color space, generating a new HSL color by selecting H, S, and L components from the source or destination according to the blend operation, and then converting the result back to RGB. In the equations below, a blended RGB color is produced according to the following pseudocode:

```

float minv3(vec3 c) {
    return min(min(c.r, c.g), c.b);
}
float maxv3(vec3 c) {
    return max(max(c.r, c.g), c.b);
}
float lumv3(vec3 c) {
    return dot(c, vec3(0.30, 0.59, 0.11));
}
float satv3(vec3 c) {
    return maxv3(c) - minv3(c);
}

// If any color components are outside [0,1], adjust the color to
// get the components in range.
vec3 ClipColor(vec3 color) {
    float lum = lumv3(color);
    float mincol = minv3(color);

```

```

float maxcol = maxv3(color);
if (mincol < 0.0) {
    color = lum + ((color-lum)*lum) / (lum-mincol);
}
if (maxcol > 1.0) {
    color = lum + ((color-lum)*(1-lum)) / (maxcol-lum);
}
return color;
}

// Take the base RGB color <cbase> and override its luminosity
// with that of the RGB color <clum>.
vec3 SetLum(vec3 cbase, vec3 clum) {
    float lbase = lumv3(cbase);
    float llum = lumv3(clum);
    float ldiff = llum - lbase;
    vec3 color = cbase + vec3(ldiff);
    return ClipColor(color);
}

// Take the base RGB color <cbase> and override its saturation with
// that of the RGB color <csat>. The override the luminosity of the
// result with that of the RGB color <clum>.
vec3 SetLumSat(vec3 cbase, vec3 csat, vec3 clum)
{
    float minbase = minv3(cbase);
    float sbase = satv3(cbase);
    float ssat = satv3(csat);
    vec3 color;
    if (sbase > 0) {
        // Equivalent (modulo rounding errors) to setting the
        // smallest (R,G,B) component to 0, the largest to <ssat>,
        // and interpolating the "middle" component based on its
        // original value relative to the smallest/largest.
        color = (cbase - minbase) * ssat / sbase;
    } else {
        color = vec3(0.0);
    }
    return SetLum(color, clum);
}

```

Table 37. Hue-Saturation-Luminosity Advanced Blend Operations

Mode	Result
VK_BLEND_OP_HSL_HUE_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f \vec{C}_s, C_d = \text{SetLumSat}(\vec{C}_s, C_d, C_d)$
VK_BLEND_OP_HSL_SATURATION_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f \vec{C}_s, C_d = \text{SetLumSat}(\vec{C}_d, C_s, C_d)$

Mode	Result
VK_BLEND_OP_HSL_COLOR_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f \vec{C}_s, C_d = \text{SetLum} \vec{C}_s, C_d$
VK_BLEND_OP_HSL_LUMINOSITY_EXT	$\vec{X}, Y, Z = \vec{1}, 1, 1$ $f \vec{C}_s, C_d = \text{SetLum} \vec{C}_d, C_s$

When using one of the operations in table [Additional RGB Blend Operations](#) as the blend operation, the source and destination colors used by these blending operations are interpreted according to `srcPremultiplied` and `dstPremultiplied`. The blending operations below are evaluated where the RGB source and destination color components are both considered to have been premultiplied by the corresponding A component.

$$\vec{R}_s', G_s', B_s' = \begin{cases} \vec{R}_s, G_s, B_s & \text{if srcPremultiplied is VK\_TRUE} \\ \vec{R}_s A_s, G_s A_s, B_s A_s & \text{if srcPremultiplied is VK\_FALSE} \end{cases}$$

$$\vec{R}_d', G_d', B_d' = \begin{cases} \vec{R}_d, G_d, B_d & \text{if dstPremultiplied is VK\_TRUE} \\ \vec{R}_d A_d, G_d A_d, B_d A_d & \text{if dstPremultiplied is VK\_FALSE} \end{cases}$$

Table 38. Additional RGB Blend Operations

Mode	Result
VK_BLEND_OP_PLUS_EXT	$\vec{R}, G, B, A = \vec{R}_s' + \vec{R}_d',$ $G_s' + G_d',$ $B_s' + B_d',$ $A_s + A_d$
VK_BLEND_OP_PLUS_CLAMPED_EXT	$\vec{R}, G, B, A = \min \vec{1}, \vec{R}_s' + \vec{R}_d',$ $\min \vec{1}, G_s' + G_d',$ $\min \vec{1}, B_s' + B_d',$ $\min \vec{1}, A_s + A_d$
VK_BLEND_OP_PLUS_CLAMPED_ALPHA_EXT	$\vec{R}, G, B, A = \min \min \vec{1}, A_s + A_d, \vec{R}_s' + \vec{R}_d',$ $\min \min \vec{1}, A_s + A_d, G_s' + G_d',$ $\min \min \vec{1}, A_s + A_d, B_s' + B_d',$ $\min \vec{1}, A_s + A_d$
VK_BLEND_OP_PLUS_DARKER_EXT	$\vec{R}, G, B, A = \max \{0, \min \vec{1}, A_s + A_d\} - \{A_s - R_s' + A_d - R_d',$ $A_s - G_s' + A_d - G_d',$ $A_s - B_s' + A_d - B_d'\},$ $\min \vec{1}, A_s + A_d$
VK_BLEND_OP_MINUS_EXT	$\vec{R}, G, B, A = \vec{R}_d' - \vec{R}_s',$ $G_d' - G_s',$ $B_d' - B_s',$ $A_d - A_s$
VK_BLEND_OP_MINUS_CLAMPED_EXT	$\vec{R}, G, B, A = \max \{0, \vec{R}_d' - \vec{R}_s',$ $G_d' - G_s',$ $B_d' - B_s',$ $\max \{0, A_d - A_s\}$

Mode	Result
VK_BLEND_OP_CONTRAST_EXT	$\begin{aligned} R_d' &= \frac{A_d}{2} + 2\left(\frac{A_d}{2} - \frac{A_d}{2} R_s'\right) - \frac{A_d}{2} R_s' - \frac{A_s}{2}, \\ G_d' &= \frac{A_d}{2} + 2\left(\frac{A_d}{2} - \frac{A_d}{2} G_s'\right) - \frac{A_d}{2} G_s' - \frac{A_s}{2}, \\ B_d' &= \frac{A_d}{2} + 2\left(\frac{A_d}{2} - \frac{A_d}{2} B_s'\right) - \frac{A_d}{2} B_s' - \frac{A_s}{2}, \\ A_d' &= A_d \end{aligned}$
VK_BLEND_OP_INVERT_OVG_EXT	$\begin{aligned} R_d' &= A_s(1 - R_d') + (1 - A_s)R_d', \\ G_d' &= A_s(1 - G_d') + (1 - A_s)G_d', \\ B_d' &= A_s(1 - B_d') + (1 - A_s)B_d', \\ A_d' &= A_s + A_d - A_s A_d \end{aligned}$
VK_BLEND_OP_RED_EXT	$R_d' = R_s', G_d' = B_d', A_d' = A_d$
VK_BLEND_OP_GREEN_EXT	$R_d' = R_d', G_d' = G_s', B_d' = A_d'$
VK_BLEND_OP_BLUE_EXT	$R_d' = R_d', G_d' = B_s', A_d' = A_d$

28.2. Logical Operations

The application **can** enable a *logical operation* between the fragment's color values and the existing value in the framebuffer attachment. This logical operation is applied prior to updating the framebuffer attachment. Logical operations are applied only for signed and unsigned integer and normalized integer framebuffers. Logical operations are not applied to floating-point or sRGB format color attachments.

Logical operations are controlled by the `logicOpEnable` and `logicOp` members of `VkPipelineColorBlendStateCreateInfo`. The `logicOp` state can also be controlled by `vkCmdSetLogicOpEXT` if graphics pipeline is created with `VK_DYNAMIC_STATE_LOGIC_OP_EXT` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. If `logicOpEnable` is `VK_TRUE`, then a logical operation selected by `logicOp` is applied between each color attachment and the fragment's corresponding output value, and blending of all attachments is treated as if it were disabled. Any attachments using color formats for which logical operations are not supported simply pass through the color values unmodified. The logical operation is applied independently for each of the red, green, blue, and alpha components. The `logicOp` is selected from the following operations:

```
// Provided by VK_VERSION_1_0
typedef enum VkLogicOp {
    VK_LOGIC_OP_CLEAR = 0,
    VK_LOGIC_OP_AND = 1,
    VK_LOGIC_OP_AND_REVERSE = 2,
    VK_LOGIC_OP_COPY = 3,
    VK_LOGIC_OP_AND_INVERTED = 4,
    VK_LOGIC_OP_NO_OP = 5,
    VK_LOGIC_OP_XOR = 6,
    VK_LOGIC_OP_OR = 7,
    VK_LOGIC_OP_NOR = 8,
    VK_LOGIC_OP_EQUIVALENT = 9,
    VK_LOGIC_OP_INVERT = 10,
    VK_LOGIC_OP_OR_REVERSE = 11,
    VK_LOGIC_OP_COPY_INVERTED = 12,
    VK_LOGIC_OP_OR_INVERTED = 13,
    VK_LOGIC_OP_NAND = 14,
    VK_LOGIC_OP_SET = 15,
} VkLogicOp;
```

The logical operations supported by Vulkan are summarized in the following table in which

- $\neg$ is bitwise invert,
- $\square$ is bitwise and,
- $\sqcup$ is bitwise or,
- $\oplus$ is bitwise exclusive or,
- s is the fragment's R_{s0} , G_{s0} , B_{s0} or A_{s0} component value for the fragment output corresponding to the color attachment being updated, and
- d is the color attachment's R, G, B or A component value:

Table 39. Logical Operations

Mode	Operation
VK_LOGIC_OP_CLEAR	0
VK_LOGIC_OP_AND	$s \square d$
VK_LOGIC_OP_AND_REVERSE	$s \square \neg d$
VK_LOGIC_OP_COPY	s
VK_LOGIC_OP_AND_INVERTED	$\neg s \square d$
VK_LOGIC_OP_NO_OP	d
VK_LOGIC_OP_XOR	$s \oplus d$
VK_LOGIC_OP_OR	$s \sqcup d$
VK_LOGIC_OP_NOR	$\neg (s \sqcup d)$
VK_LOGIC_OP_EQUIVALENT	$\neg (s \square d)$
VK_LOGIC_OP_INVERT	$\neg d$
VK_LOGIC_OP_OR_REVERSE	$s \sqcup \neg d$
VK_LOGIC_OP_COPY_INVERTED	$\neg s$
VK_LOGIC_OP_OR_INVERTED	$\neg s \sqcup d$
VK_LOGIC_OP_NAND	$\neg (s \square d)$
VK_LOGIC_OP_SET	all 1s

The result of the logical operation is then written to the color attachment as controlled by the component write mask, described in [Blend Operations](#).

To [dynamically set](#) the logical operation to apply for blend state, call:

```
// Provided by VK_EXT_extended_dynamic_state2
void vkCmdSetLogicOpEXT(
    VkCommandBuffer          commandBuffer,
    VkLogicOp                 logicOp);
```


- `commandBuffer` is the command buffer into which the command will be recorded.
- `logicOp` specifies the logical operation to apply for blend state.

This command sets the logical operation for blend state for subsequent drawing commands when the graphics pipeline is created with `VK_DYNAMIC_STATE_LOGIC_OP_EXT` set in `VkPipelineDynamicStateCreateInfo::pDynamicStates`. Otherwise, this state is specified by the `VkPipelineColorBlendStateCreateInfo::logicOp` value used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetLogicOpEXT-None-09422
At least one of the following **must** be true:
 - The `extendedDynamicState2LogicOp` feature is enabled

Valid Usage (Implicit)

- VUID-vkCmdSetLogicOpEXT-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdSetLogicOpEXT-logicOp-parameter
`logicOp` **must** be a valid `VkLogicOp` value
- VUID-vkCmdSetLogicOpEXT-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdSetLogicOpEXT-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_GRAPHICS_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

28.3. Color Write Mask

Bits which **can** be set in `VkPipelineColorBlendAttachmentState::colorWriteMask`, determining whether the final color values R, G, B and A are written to the framebuffer attachment, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkColorComponentFlagBits {
    VK_COLOR_COMPONENT_R_BIT = 0x00000001,
    VK_COLOR_COMPONENT_G_BIT = 0x00000002,
    VK_COLOR_COMPONENT_B_BIT = 0x00000004,
    VK_COLOR_COMPONENT_A_BIT = 0x00000008,
} VkColorComponentFlagBits;
```

- `VK_COLOR_COMPONENT_R_BIT` specifies that the R value is written to the color attachment for the appropriate sample. Otherwise, the value in memory is unmodified.
- `VK_COLOR_COMPONENT_G_BIT` specifies that the G value is written to the color attachment for the appropriate sample. Otherwise, the value in memory is unmodified.
- `VK_COLOR_COMPONENT_B_BIT` specifies that the B value is written to the color attachment for the appropriate sample. Otherwise, the value in memory is unmodified.
- `VK_COLOR_COMPONENT_A_BIT` specifies that the A value is written to the color attachment for the appropriate sample. Otherwise, the value in memory is unmodified.

The color write mask operation is applied regardless of whether blending is enabled.

The color write mask operation is applied only if `Color Write Enable` is enabled for the respective attachment. Otherwise the color write mask is ignored and writes to all components of the attachment are disabled.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkColorComponentFlags;
```

`VkColorComponentFlags` is a bitmask type for setting a mask of zero or more `VkColorComponentFlagBits`.

28.4. Color Write Enable

The `VkPipelineColorWriteCreateInfoEXT` structure is defined as:

```
// Provided by VK_EXT_color_write_enable
typedef struct VkPipelineColorWriteCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           attachmentCount;
    const VkBool32*    pColorWriteEnables;
} VkPipelineColorWriteCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `attachmentCount` is the number of `VkBool32` elements in `pColorWriteEnables`.
- `pColorWriteEnables` is a pointer to an array of per target attachment boolean values specifying whether color writes are enabled for the given attachment.

When this structure is included in the `pNext` chain of `VkPipelineColorBlendStateCreateInfo`, it defines per-attachment color write state. If this structure is not included in the `pNext` chain, it is equivalent to specifying this structure with `attachmentCount` equal to the `attachmentCount` member of `VkPipelineColorBlendStateCreateInfo`, and `pColorWriteEnables` pointing to an array of as many `VK_TRUE` values.

If the `colorWriteEnable` feature is not enabled, all `VkBool32` elements in the `pColorWriteEnables` array **must** be `VK_TRUE`.

Color Write Enable interacts with the [Color Write Mask](#) as follows:

- If `colorWriteEnable` is `VK_TRUE`, writes to the attachment are determined by the `colorWriteMask`.
- If `colorWriteEnable` is `VK_FALSE`, the `colorWriteMask` is ignored and writes to all components of the attachment are disabled. This is equivalent to specifying a `colorWriteMask` of 0.

Valid Usage

- VUID-VkPipelineColorWriteCreateInfoEXT-pAttachments-04801
If the `colorWriteEnable` feature is not enabled, all elements of `pColorWriteEnables` **must** be `VK_TRUE`
- VUID-VkPipelineColorWriteCreateInfoEXT-attachmentCount-07608
`attachmentCount` **must** be equal to the `attachmentCount` member of the `VkPipelineColorBlendStateCreateInfo` structure specified during pipeline creation
- VUID-VkPipelineColorWriteCreateInfoEXT-attachmentCount-06655
`attachmentCount` **must** be less than or equal to the `maxColorAttachments` member of `VkPhysicalDeviceLimits`

Valid Usage (Implicit)

- VUID-VkPipelineColorWriteCreateInfoEXT-sType-sType

sType **must** be VK_STRUCTURE_TYPE_PIPELINE_COLOR_WRITE_CREATE_INFO_EXT

- VUID-VkPipelineColorWriteCreateInfoEXT-pColorWriteEnables-parameter
If attachmentCount is not 0, pColorWriteEnables **must** be a valid pointer to an array of attachmentCount VkBool32 values

To dynamically enable or disable writes to a color attachment, call:

```
// Provided by VK_EXT_color_write_enable
void vkCmdSetColorWriteEnableEXT(
    VkCommandBuffer      commandBuffer,
    uint32_t             attachmentCount,
    const VkBool32*       pColorWriteEnables);
```

- commandBuffer is the command buffer into which the command will be recorded.
- attachmentCount is the number of VkBool32 elements in pColorWriteEnables.
- pColorWriteEnables is a pointer to an array of per target attachment boolean values specifying whether color writes are enabled for the given attachment.

This command sets the color write enables for subsequent drawing commands when the graphics pipeline is created with VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT set in VkPipelineDynamicStateCreateInfo::pDynamicStates. Otherwise, this state is specified by the VkPipelineColorWriteCreateInfoEXT::pColorWriteEnables values used to create the currently active pipeline.

Valid Usage

- VUID-vkCmdSetColorWriteEnableEXT-None-04803
The colorWriteEnable feature **must** be enabled
- VUID-vkCmdSetColorWriteEnableEXT-attachmentCount-06656
attachmentCount **must** be less than or equal to the maxColorAttachments member of VkPhysicalDeviceLimits

Valid Usage (Implicit)

- VUID-vkCmdSetColorWriteEnableEXT-commandBuffer-parameter
commandBuffer **must** be a valid VkCommandBuffer handle
- VUID-vkCmdSetColorWriteEnableEXT-pColorWriteEnables-parameter
pColorWriteEnables **must** be a valid pointer to an array of attachmentCount VkBool32 values
- VUID-vkCmdSetColorWriteEnableEXT-commandBuffer-recording
commandBuffer **must** be in the recording state
- VUID-vkCmdSetColorWriteEnableEXT-commandBuffer-cmdpool
The VkCommandPool that commandBuffer was allocated from **must** support

VK_QUEUE_GRAPHICS_BIT operations

- VUID-vkCmdSetColorWriteEnableEXT-attachmentCount-arraylength
attachmentCount **must** be greater than 0

Host Synchronization

- Host access to commandBuffer **must** be externally synchronized
- Host access to the VkCommandPool that commandBuffer was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_GRAPHICS_BIT	State

Chapter 29. Dispatching Commands

The *dispatching commands* described in this chapter provoke work in a compute pipeline. Dispatching commands are recorded into a command buffer and when executed by a queue, will produce work which executes according to the bound compute pipeline. A compute pipeline **must** be bound to a command buffer before any dispatching commands are recorded in that command buffer.

To record a dispatch, call:

```
// Provided by VK_VERSION_1_0
void vkCmdDispatch(
    VkCommandBuffer          commandBuffer,
    uint32_t                 groupCountX,
    uint32_t                 groupCountY,
    uint32_t                 groupCountZ);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `groupCountX` is the number of local workgroups to dispatch in the X dimension.
- `groupCountY` is the number of local workgroups to dispatch in the Y dimension.
- `groupCountZ` is the number of local workgroups to dispatch in the Z dimension.

When the command is executed, a global workgroup consisting of `groupCountX` × `groupCountY` × `groupCountZ` local workgroups is assembled.

Valid Usage

- VUID-vkCmdDispatch-magFilter-04553
If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdDispatch-magFilter-09598
If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- VUID-vkCmdDispatch-mipmapMode-04770
If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDispatch-mipmapMode-09599
If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- VUID-vkCmdDispatch-unnormlizedCoordinates-09635
If a `VkSampler` created with `unnormlizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `levelCount` and `layerCount` **must** be 1
- VUID-vkCmdDispatch-None-08609
If a `VkSampler` created with `unnormlizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `viewType` **must** be `VK_IMAGE_VIEW_TYPE_1D` or `VK_IMAGE_VIEW_TYPE_2D`
- VUID-vkCmdDispatch-None-08610
If a `VkSampler` created with `unnormlizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name
- VUID-vkCmdDispatch-None-08611
If a `VkSampler` created with `unnormlizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values
- VUID-vkCmdDispatch-aspectMask-06478
If a `VkImageView` is sampled with `depth comparison`, the image view **must** have been created with an `aspectMask` that contains `VK_IMAGE_ASPECT_DEPTH_BIT`
- VUID-vkCmdDispatch-None-02691
If a `VkImageView` is accessed using atomic operations as a result of this command, then the image view's `format features` **must** contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT`
- VUID-vkCmdDispatch-None-07888
If a `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's `format features` **must** contain `VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT`
- VUID-vkCmdDispatch-None-02692
If a `VkImageView` is sampled with `VK_FILTER_CUBIC_EXT` as a result of this command, then the image view's `format features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT`
- VUID-vkCmdDispatch-None-02693
If the `VK_EXT_filter_cubic` extension is not enabled and any `VkImageView` is sampled with `VK_FILTER_CUBIC_EXT` as a result of this command, it **must** not have a `VkImageViewType` of `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`
- VUID-vkCmdDispatch-filterCubic-02694

Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)

- VUID-vkCmdDispatch-filterCubicMinmax-02695

Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) with a reduction mode of either [VK_SAMPLER_REDUCTION_MODE_MIN](#) or [VK_SAMPLER_REDUCTION_MODE_MAX](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering together with minmax filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)

- VUID-vkCmdDispatch-None-08600

For each set n that is statically used by a [bound shader](#), a descriptor set **must** have been bound to n at the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for set n , with the [VkPipelineLayout](#) used to create the current [VkPipeline](#), as described in [Pipeline Layout Compatibility](#)

- VUID-vkCmdDispatch-None-08601

For each push constant that is statically used by a [bound shader](#), a push constant value **must** have been set for the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for push constants, with the [VkPipelineLayout](#) used to create the current [VkPipeline](#), as described in [Pipeline Layout Compatibility](#)

- VUID-vkCmdDispatch-None-10068

For each array of resources that is used by a [bound shader](#), the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command

- VUID-vkCmdDispatch-None-08114

Descriptors in each bound descriptor set, specified via [vkCmdBindDescriptorSets](#), **must** be valid as described by [descriptor validity](#) if they are statically used by a [bound shader](#)

- VUID-vkCmdDispatch-imageLayout-00344

If an image descriptor is accessed by a shader, the [VkImageLayout](#) **must** match the subresource accessible from the [VkImageView](#) as defined by the [image layout matching rules](#)

- VUID-vkCmdDispatch-None-08606

A valid pipeline **must** be bound to the pipeline bind point used by this command

- VUID-vkCmdDispatch-None-08608

There **must** not have been any calls to dynamic state setting commands for any state specified statically in the [VkPipeline](#) object bound to the pipeline bind point used by this command, since that pipeline was bound

- VUID-vkCmdDispatch-uniformBuffers-06935

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a uniform buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatch-storageBuffers-06936

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a storage buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatch-commandBuffer-02707

If [commandBuffer](#) is an unprotected command buffer and [protectedNoFault](#) is not supported, any resource accessed by [bound shaders](#) **must** not be a protected resource

- VUID-vkCmdDispatch-viewType-07752

If a [VkImageView](#) is accessed as a result of this command, then the image view's [viewType](#) **must** match the [Dim](#) operand of the [OpTypeImage](#) as described in [Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types](#)

- VUID-vkCmdDispatch-format-07753

If a [VkImageView](#) or [VkBufferView](#) is accessed as a result of this command, then the [numeric type](#) of the view's [format](#) and the [Sampled Type](#) operand of the [OpTypeImage](#) **must** match

- VUID-vkCmdDispatch-OpImageWrite-08795

If a [VkImageView](#) is accessed using [OpImageWrite](#) as a result of this command, then the [Type](#) of the [Texel](#) operand of that instruction **must** have at least as many components as the image view's format

- VUID-vkCmdDispatch-OpImageWrite-04469

If a [VkBufferView](#) is accessed using [OpImageWrite](#) as a result of this command, then the [Type](#) of the [Texel](#) operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDispatch-SampledType-04470

If a [VkImageView](#) with a [VkFormat](#) that has a 64-bit component width is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 64

- VUID-vkCmdDispatch-SampledType-04471

If a [VkImageView](#) with a [VkFormat](#) that has a component width less than 64-bit is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 32

- VUID-vkCmdDispatch-SampledType-04472

If a [VkBufferView](#) with a [VkFormat](#) that has a 64-bit component width is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 64

- VUID-vkCmdDispatch-SampledType-04473

If a [VkBufferView](#) with a [VkFormat](#) that has a component width less than 64-bit is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 32

- VUID-vkCmdDispatch-sparseImageInt64Atomics-04474

If the [sparseImageInt64Atomics](#) feature is not enabled, [VkImage](#) objects created with the [VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT](#) flag **must** not be accessed by atomic instructions through an [OpTypeImage](#) with a [SampledType](#) with a [Width](#) of 64 by this command

- VUID-vkCmdDispatch-sparseImageInt64Atomics-04475
If the `sparseImageInt64Atomics` feature is not enabled, `VkBuffer` objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDispatch-None-07288
Any shader invocation executed by this command **must** `terminate`
- VUID-vkCmdDispatch-None-09600
If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, all image subresources identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDispatch-None-10672
this command **must** be called outside of a render pass instance
- VUID-vkCmdDispatch-commandBuffer-02712
If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, any resource written to by the `VkPipeline` object bound to the pipeline bind point used by this command **must** not be an unprotected resource
- VUID-vkCmdDispatch-commandBuffer-02713
If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, pipeline stages other than the framebuffer-space and compute stages in the `VkPipeline` object bound to the pipeline bind point used by this command **must** not write to any resource
- VUID-vkCmdDispatch-groupCountX-00386
`groupCountX` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[0]`
- VUID-vkCmdDispatch-groupCountY-00387
`groupCountY` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[1]`
- VUID-vkCmdDispatch-groupCountZ-00388
`groupCountZ` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[2]`

Valid Usage (Implicit)

- VUID-vkCmdDispatch-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdDispatch-commandBuffer-recording
`commandBuffer` **must** be in the `recording` state
- VUID-vkCmdDispatch-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT	Action

To record an indirect dispatching command, call:

```
// Provided by VK_VERSION_1_0
void vkCmdDispatchIndirect(
    VkCommandBuffer          commandBuffer,
    VkBuffer                 buffer,
    VkDeviceSize             offset);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `buffer` is the buffer containing dispatch parameters.
- `offset` is the byte offset into `buffer` where parameters begin.

`vkCmdDispatchIndirect` behaves similarly to `vkCmdDispatch` except that the parameters are read by the device from a buffer during execution. The parameters of the dispatch are encoded in a `VkDispatchIndirectCommand` structure taken from `buffer` starting at `offset`.

Valid Usage

- VUID-vkCmdDispatchIndirect-magFilter-04553
If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdDispatchIndirect-magFilter-09598
If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format features` **must** contain

VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT

- VUID-vkCmdDispatchIndirect-mipmapMode-04770

If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDispatchIndirect-mipmapMode-09599

If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`

- VUID-vkCmdDispatchIndirect-unnormalizedCoordinates-09635

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `levelCount` and `layerCount` **must** be 1

- VUID-vkCmdDispatchIndirect-None-08609

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `viewType` **must** be `VK_IMAGE_VIEW_TYPE_1D` or `VK_IMAGE_VIEW_TYPE_2D`

- VUID-vkCmdDispatchIndirect-None-08610

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name

- VUID-vkCmdDispatchIndirect-None-08611

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values

- VUID-vkCmdDispatchIndirect-aspectMask-06478

If a `VkImageView` is sampled with `depth comparison`, the image view **must** have been created with an `aspectMask` that contains `VK_IMAGE_ASPECT_DEPTH_BIT`

- VUID-vkCmdDispatchIndirect-None-02691

If a `VkImageView` is accessed using atomic operations as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT`

- VUID-vkCmdDispatchIndirect-None-07888

If a `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's `format` `features` **must** contain `VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT`

- VUID-vkCmdDispatchIndirect-None-02692

If a `VkImageView` is sampled with `VK_FILTER_CUBIC_EXT` as a result of this command, then

the image view's format features **must** contain
`VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT`

- VUID-vkCmdDispatchIndirect-None-02693
If the `VK_EXT_filter_cubic` extension is not enabled and any `VkImageView` is sampled with `VK_FILTER_CUBIC_EXT` as a result of this command, it **must** not have a `VkImageViewType` of `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`
- VUID-vkCmdDispatchIndirect-filterCubic-02694
Any `VkImageView` being sampled with `VK_FILTER_CUBIC_EXT` as a result of this command **must** have a `VkImageViewType` and format that supports cubic filtering, as specified by `VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic` returned by `vkGetPhysicalDeviceImageFormatProperties2`
- VUID-vkCmdDispatchIndirect-filterCubicMinmax-02695
Any `VkImageView` being sampled with `VK_FILTER_CUBIC_EXT` with a reduction mode of either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` as a result of this command **must** have a `VkImageViewType` and format that supports cubic filtering together with minmax filtering, as specified by `VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax` returned by `vkGetPhysicalDeviceImageFormatProperties2`
- VUID-vkCmdDispatchIndirect-None-08600
For each set n that is statically used by a `bound shader`, a descriptor set **must** have been bound to n at the same pipeline bind point, with a `VkPipelineLayout` that is compatible for set n , with the `VkPipelineLayout` used to create the current `VkPipeline`, as described in [Pipeline Layout Compatibility](#)
- VUID-vkCmdDispatchIndirect-None-08601
For each push constant that is statically used by a `bound shader`, a push constant value **must** have been set for the same pipeline bind point, with a `VkPipelineLayout` that is compatible for push constants, with the `VkPipelineLayout` used to create the current `VkPipeline`, as described in [Pipeline Layout Compatibility](#)
- VUID-vkCmdDispatchIndirect-None-10068
For each array of resources that is used by a `bound shader`, the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command
- VUID-vkCmdDispatchIndirect-None-08114
Descriptors in each bound descriptor set, specified via `vkCmdBindDescriptorSets`, **must** be valid as described by [descriptor validity](#) if they are statically used by a `bound shader`
- VUID-vkCmdDispatchIndirect-imageLayout-00344
If an image descriptor is accessed by a shader, the `VkImageLayout` **must** match the subresource accessible from the `VkImageView` as defined by the [image layout matching rules](#)
- VUID-vkCmdDispatchIndirect-None-08606
A valid pipeline **must** be bound to the pipeline bind point used by this command
- VUID-vkCmdDispatchIndirect-None-08608
There **must** not have been any calls to dynamic state setting commands for any state specified statically in the `VkPipeline` object bound to the pipeline bind point used by this

command, since that pipeline was bound

- VUID-vkCmdDispatchIndirect-uniformBuffers-06935

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a uniform buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatchIndirect-storageBuffers-06936

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this command accesses a storage buffer, and the [robustBufferAccess](#) feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatchIndirect-commandBuffer-02707

If [commandBuffer](#) is an unprotected command buffer and [protectedNoFault](#) is not supported, any resource accessed by [bound shaders](#) **must** not be a protected resource

- VUID-vkCmdDispatchIndirect-viewType-07752

If a [VkImageView](#) is accessed as a result of this command, then the image view's [viewType](#) **must** match the [Dim](#) operand of the [OpTypeImage](#) as described in [Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types](#)

- VUID-vkCmdDispatchIndirect-format-07753

If a [VkImageView](#) or [VkBufferView](#) is accessed as a result of this command, then the [numeric type](#) of the view's [format](#) and the [Sampled Type](#) operand of the [OpTypeImage](#) **must** match

- VUID-vkCmdDispatchIndirect-OpImageWrite-08795

If a [VkImageView](#) is accessed using [OpImageWrite](#) as a result of this command, then the [Type](#) of the [Texel](#) operand of that instruction **must** have at least as many components as the image view's format

- VUID-vkCmdDispatchIndirect-OpImageWrite-04469

If a [VkBufferView](#) is accessed using [OpImageWrite](#) as a result of this command, then the [Type](#) of the [Texel](#) operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDispatchIndirect-SampledType-04470

If a [VkImageView](#) with a [VkFormat](#) that has a 64-bit component width is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 64

- VUID-vkCmdDispatchIndirect-SampledType-04471

If a [VkImageView](#) with a [VkFormat](#) that has a component width less than 64-bit is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 32

- VUID-vkCmdDispatchIndirect-SampledType-04472

If a [VkBufferView](#) with a [VkFormat](#) that has a 64-bit component width is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 64

- VUID-vkCmdDispatchIndirect-SampledType-04473

If a [VkBufferView](#) with a [VkFormat](#) that has a component width less than 64-bit is accessed as a result of this command, the [SampledType](#) of the [OpTypeImage](#) operand of that instruction **must** have a [Width](#) of 32

- VUID-vkCmdDispatchIndirect-sparseImageInt64Atomics-04474

If the [sparseImageInt64Atomics](#) feature is not enabled, [VkImage](#) objects created with the [VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT](#) flag **must** not be accessed by atomic instructions through an [OpTypeImage](#) with a [SampledType](#) with a [Width](#) of 64 by this command

- VUID-vkCmdDispatchIndirect-sparseImageInt64Atomics-04475

If the [sparseImageInt64Atomics](#) feature is not enabled, [VkBuffer](#) objects created with the [VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT](#) flag **must** not be accessed by atomic instructions through an [OpTypeImage](#) with a [SampledType](#) with a [Width](#) of 64 by this command

- VUID-vkCmdDispatchIndirect-None-07288

Any shader invocation executed by this command **must** [terminate](#)

- VUID-vkCmdDispatchIndirect-None-09600

If a descriptor with type equal to any of [VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE](#), [VK_DESCRIPTOR_TYPE_STORAGE_IMAGE](#), or [VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT](#) is accessed as a result of this command, all image subresources identified by that descriptor **must** be in the image layout identified when the descriptor was written

- VUID-vkCmdDispatchIndirect-None-10672

this command **must** be called outside of a render pass instance

- VUID-vkCmdDispatchIndirect-buffer-02708

If [buffer](#) is non-sparse then it **must** be bound completely and contiguously to a single [VkDeviceMemory](#) object

- VUID-vkCmdDispatchIndirect-buffer-02709

[buffer](#) **must** have been created with the [VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT](#) bit set

- VUID-vkCmdDispatchIndirect-offset-02710

[offset](#) **must** be a multiple of 4

- VUID-vkCmdDispatchIndirect-commandBuffer-02711

[commandBuffer](#) **must** not be a protected command buffer

- VUID-vkCmdDispatchIndirect-offset-00407

The sum of [offset](#) and the size of [VkDispatchIndirectCommand](#) **must** be less than or equal to the size of [buffer](#)

Valid Usage (Implicit)

- VUID-vkCmdDispatchIndirect-commandBuffer-parameter

[commandBuffer](#) **must** be a valid [VkCommandBuffer](#) handle

- VUID-vkCmdDispatchIndirect-buffer-parameter

[buffer](#) **must** be a valid [VkBuffer](#) handle

- VUID-vkCmdDispatchIndirect-commandBuffer-recording

[commandBuffer](#) **must** be in the [recording state](#)

- VUID-vkCmdDispatchIndirect-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT` operations
- VUID-vkCmdDispatchIndirect-commonparent
Both of `buffer`, and `commandBuffer` **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT	Action

The `VkDispatchIndirectCommand` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkDispatchIndirectCommand {
    uint32_t    x;
    uint32_t    y;
    uint32_t    z;
} VkDispatchIndirectCommand;
```

- `x` is the number of local workgroups to dispatch in the X dimension.
- `y` is the number of local workgroups to dispatch in the Y dimension.
- `z` is the number of local workgroups to dispatch in the Z dimension.

The members of `VkDispatchIndirectCommand` have the same meaning as the corresponding parameters of `vkCmdDispatch`.

Valid Usage

- VUID-VkDispatchIndirectCommand-x-00417
`x` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[0]`
- VUID-VkDispatchIndirectCommand-y-00418
`y` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[1]`

- VUID-VkDispatchIndirectCommand-z-00419
`z` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[2]`

To record a dispatch using non-zero base values for the components of `WorkgroupId`, call:

```
// Provided by VK_VERSION_1_1
void vkCmdDispatchBase(
    VkCommandBuffer          commandBuffer,
    uint32_t                 baseGroupX,
    uint32_t                 baseGroupY,
    uint32_t                 baseGroupZ,
    uint32_t                 groupCountX,
    uint32_t                 groupCountY,
    uint32_t                 groupCountZ);
```

- `commandBuffer` is the command buffer into which the command will be recorded.
- `baseGroupX` is the start value for the X component of `WorkgroupId`.
- `baseGroupY` is the start value for the Y component of `WorkgroupId`.
- `baseGroupZ` is the start value for the Z component of `WorkgroupId`.
- `groupCountX` is the number of local workgroups to dispatch in the X dimension.
- `groupCountY` is the number of local workgroups to dispatch in the Y dimension.
- `groupCountZ` is the number of local workgroups to dispatch in the Z dimension.

When the command is executed, a global workgroup consisting of `groupCountX × groupCountY × groupCountZ` local workgroups is assembled, with `WorkgroupId` values ranging from [`baseGroup*`, `baseGroup* + groupCount*`) in each component. `vkCmdDispatch` is equivalent to `vkCmdDispatchBase(0,0,0,groupCountX,groupCountY,groupCountZ)`.

Valid Usage

- VUID-vkCmdDispatchBase-magFilter-04553
 If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR`, `reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- VUID-vkCmdDispatchBase-magFilter-09598
 If a `VkSampler` created with `magFilter` or `minFilter` equal to `VK_FILTER_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- VUID-vkCmdDispatchBase-mipmapMode-04770
 If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR`,

`reductionMode` equal to `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`, and `compareEnable` equal to `VK_FALSE` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

- VUID-vkCmdDispatchBase-mipmapMode-09599

If a `VkSampler` created with `mipmapMode` equal to `VK_SAMPLER_MIPMAP_MODE_LINEAR` and `reductionMode` equal to either `VK_SAMPLER_REDUCTION_MODE_MIN` or `VK_SAMPLER_REDUCTION_MODE_MAX` is used to sample a `VkImageView` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`

- VUID-vkCmdDispatchBase-unnormalizedCoordinates-09635

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `levelCount` and `layerCount` **must** be 1

- VUID-vkCmdDispatchBase-None-08609

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the image view's `viewType` **must** be `VK_IMAGE_VIEW_TYPE_1D` or `VK_IMAGE_VIEW_TYPE_2D`

- VUID-vkCmdDispatchBase-None-08610

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name

- VUID-vkCmdDispatchBase-None-08611

If a `VkSampler` created with `unnormalizedCoordinates` equal to `VK_TRUE` is used to sample a `VkImageView` as a result of this command, then the sampler **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values

- VUID-vkCmdDispatchBase-aspectMask-06478

If a `VkImageView` is sampled with `depth comparison`, the image view **must** have been created with an `aspectMask` that contains `VK_IMAGE_ASPECT_DEPTH_BIT`

- VUID-vkCmdDispatchBase-None-02691

If a `VkImageView` is accessed using atomic operations as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT`

- VUID-vkCmdDispatchBase-None-07888

If a `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` descriptor is accessed using atomic operations as a result of this command, then the storage texel buffer's `format` `features` **must** contain `VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT`

- VUID-vkCmdDispatchBase-None-02692

If a `VkImageView` is sampled with `VK_FILTER_CUBIC_EXT` as a result of this command, then the image view's `format` `features` **must** contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT`

- VUID-vkCmdDispatchBase-None-02693

If the [VK_EXT_filter_cubic](#) extension is not enabled and any [VkImageView](#) is sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command, it **must** not have a [VkImageViewType](#) of [VK_IMAGE_VIEW_TYPE_3D](#), [VK_IMAGE_VIEW_TYPE_CUBE](#), or [VK_IMAGE_VIEW_TYPE_CUBE_ARRAY](#)

- VUID-vkCmdDispatchBase-filterCubic-02694

Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubic](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)

- VUID-vkCmdDispatchBase-filterCubicMinmax-02695

Any [VkImageView](#) being sampled with [VK_FILTER_CUBIC_EXT](#) with a reduction mode of either [VK_SAMPLER_REDUCTION_MODE_MIN](#) or [VK_SAMPLER_REDUCTION_MODE_MAX](#) as a result of this command **must** have a [VkImageViewType](#) and format that supports cubic filtering together with minmax filtering, as specified by [VkFilterCubicImageViewImageFormatPropertiesEXT::filterCubicMinmax](#) returned by [vkGetPhysicalDeviceImageFormatProperties2](#)

- VUID-vkCmdDispatchBase-None-08600

For each set n that is statically used by a [bound shader](#), a descriptor set **must** have been bound to n at the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for set n , with the [VkPipelineLayout](#) used to create the current [VkPipeline](#), as described in [Pipeline Layout Compatibility](#)

- VUID-vkCmdDispatchBase-None-08601

For each push constant that is statically used by a [bound shader](#), a push constant value **must** have been set for the same pipeline bind point, with a [VkPipelineLayout](#) that is compatible for push constants, with the [VkPipelineLayout](#) used to create the current [VkPipeline](#), as described in [Pipeline Layout Compatibility](#)

- VUID-vkCmdDispatchBase-None-10068

For each array of resources that is used by a [bound shader](#), the indices used to access members of the array **must** be less than the descriptor count for the identified binding in the descriptor sets used by this command

- VUID-vkCmdDispatchBase-None-08114

Descriptors in each bound descriptor set, specified via [vkCmdBindDescriptorSets](#), **must** be valid as described by [descriptor validity](#) if they are statically used by a [bound shader](#)

- VUID-vkCmdDispatchBase-imageLayout-00344

If an image descriptor is accessed by a shader, the [VkImageLayout](#) **must** match the subresource accessible from the [VkImageView](#) as defined by the [image layout matching rules](#)

- VUID-vkCmdDispatchBase-None-08606

A valid pipeline **must** be bound to the pipeline bind point used by this command

- VUID-vkCmdDispatchBase-None-08608

There **must** not have been any calls to dynamic state setting commands for any state specified statically in the [VkPipeline](#) object bound to the pipeline bind point used by this command, since that pipeline was bound

- VUID-vkCmdDispatchBase-uniformBuffers-06935

If any stage of the [VkPipeline](#) object bound to the pipeline bind point used by this

command accesses a uniform buffer, and the **robustBufferAccess** feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatchBase-storageBuffers-06936

If any stage of the **VkPipeline** object bound to the pipeline bind point used by this command accesses a storage buffer, and the **robustBufferAccess** feature is not enabled, that stage **must** not access values outside of the range of the buffer as specified in the descriptor set bound to the same pipeline bind point

- VUID-vkCmdDispatchBase-commandBuffer-02707

If **commandBuffer** is an unprotected command buffer and **protectedNoFault** is not supported, any resource accessed by **bound shaders** **must** not be a protected resource

- VUID-vkCmdDispatchBase-viewType-07752

If a **VkImageView** is accessed as a result of this command, then the image view's **viewType** **must** match the **Dim** operand of the **OpTypeImage** as described in [Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types](#)

- VUID-vkCmdDispatchBase-format-07753

If a **VkImageView** or **VkBufferView** is accessed as a result of this command, then the **numeric type** of the view's **format** and the **Sampled Type** operand of the **OpTypeImage** **must** match

- VUID-vkCmdDispatchBase-OpImageWrite-08795

If a **VkImageView** is accessed using **OpImageWrite** as a result of this command, then the **Type** of the **Texel** operand of that instruction **must** have at least as many components as the image view's format

- VUID-vkCmdDispatchBase-OpImageWrite-04469

If a **VkBufferView** is accessed using **OpImageWrite** as a result of this command, then the **Type** of the **Texel** operand of that instruction **must** have at least as many components as the buffer view's format

- VUID-vkCmdDispatchBase-SampledType-04470

If a **VkImageView** with a **VkFormat** that has a 64-bit component width is accessed as a result of this command, the **SampledType** of the **OpTypeImage** operand of that instruction **must** have a **Width** of 64

- VUID-vkCmdDispatchBase-SampledType-04471

If a **VkImageView** with a **VkFormat** that has a component width less than 64-bit is accessed as a result of this command, the **SampledType** of the **OpTypeImage** operand of that instruction **must** have a **Width** of 32

- VUID-vkCmdDispatchBase-SampledType-04472

If a **VkBufferView** with a **VkFormat** that has a 64-bit component width is accessed as a result of this command, the **SampledType** of the **OpTypeImage** operand of that instruction **must** have a **Width** of 64

- VUID-vkCmdDispatchBase-SampledType-04473

If a **VkBufferView** with a **VkFormat** that has a component width less than 64-bit is accessed as a result of this command, the **SampledType** of the **OpTypeImage** operand of that instruction **must** have a **Width** of 32

- VUID-vkCmdDispatchBase-sparseImageInt64Atomics-04474
If the `sparseImageInt64Atomics` feature is not enabled, `VkImage` objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDispatchBase-sparseImageInt64Atomics-04475
If the `sparseImageInt64Atomics` feature is not enabled, `VkBuffer` objects created with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag **must** not be accessed by atomic instructions through an `OpTypeImage` with a `SampledType` with a `Width` of 64 by this command
- VUID-vkCmdDispatchBase-None-07288
Any shader invocation executed by this command **must** `terminate`
- VUID-vkCmdDispatchBase-None-09600
If a descriptor with type equal to any of `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` is accessed as a result of this command, all image subresources identified by that descriptor **must** be in the image layout identified when the descriptor was written
- VUID-vkCmdDispatchBase-None-10672
this command **must** be called outside of a render pass instance
- VUID-vkCmdDispatchBase-commandBuffer-02712
If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, any resource written to by the `VkPipeline` object bound to the pipeline bind point used by this command **must** not be an unprotected resource
- VUID-vkCmdDispatchBase-commandBuffer-02713
If `commandBuffer` is a protected command buffer and `protectedNoFault` is not supported, pipeline stages other than the framebuffer-space and compute stages in the `VkPipeline` object bound to the pipeline bind point used by this command **must** not write to any resource
- VUID-vkCmdDispatchBase-baseGroupX-00421
`baseGroupX` **must** be less than `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[0]`
- VUID-vkCmdDispatchBase-baseGroupY-00422
`baseGroupY` **must** be less than `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[1]`
- VUID-vkCmdDispatchBase-baseGroupZ-00423
`baseGroupZ` **must** be less than `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[2]`
- VUID-vkCmdDispatchBase-groupCountX-00424
`groupCountX` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[0]` minus `baseGroupX`
- VUID-vkCmdDispatchBase-groupCountY-00425
`groupCountY` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[1]` minus `baseGroupY`
- VUID-vkCmdDispatchBase-groupCountZ-00426
`groupCountZ` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupCount[2]` minus `baseGroupZ`
- VUID-vkCmdDispatchBase-baseGroupX-00427

If any of `baseGroupX`, `baseGroupY`, or `baseGroupZ` are not zero, then the bound compute pipeline **must** have been created with the `VK_PIPELINE_CREATE_DISPATCH_BASE_BIT` flag

Valid Usage (Implicit)

- VUID-vkCmdDispatchBase-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdDispatchBase-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdDispatchBase-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT	Action

Chapter 30. Sparse Resources

As documented in [Resource Memory Association](#), [VkBuffer](#) and [VkImage](#) resources in Vulkan **must** be bound completely and contiguously to a single [VkDeviceMemory](#) object. This binding **must** be done before the resource is used, and the binding is immutable for the lifetime of the resource.

Sparse resources relax these restrictions and provide these additional features:

- Sparse resources **can** be bound non-contiguously to one or more [VkDeviceMemory](#) allocations.
- Sparse resources **can** be re-bound to different memory allocations over the lifetime of the resource.
- Sparse resources **can** have descriptors generated and used orthogonally with memory binding commands.

Sparse resources are not supported in Vulkan SC, due to complexity and the necessity of being able to update page table mappings at runtime [\[SCID-8\]](#). However, the sparse resource features, properties, resource creation flags, and definitions have been retained for completeness and compatibility.

All sparse resource [physical device features](#) **must** not be advertised as supported, and the related [physical device sparse properties](#) and [physical device limits](#) **must** be reported accordingly.

30.1. Sparse Resource Features

Sparse resources have several features that **must** be enabled explicitly at resource creation time. The features are enabled by including bits in the [flags](#) parameter of [VkImageCreateInfo](#) or [VkBufferCreateInfo](#). Each feature also has one or more corresponding feature enables specified in [VkPhysicalDeviceFeatures](#).

- The [sparseBinding](#) feature is the base, and provides the following capabilities:
 - Resources **can** be bound at some defined (sparse block) granularity.
 - The entire resource **must** be bound to memory before use regardless of regions actually accessed.
 - No specific mapping of image region to memory offset is defined, i.e. the location that each texel corresponds to in memory is implementation-dependent.
 - Sparse buffers have a well-defined mapping of buffer range to memory range, where an offset into a range of the buffer that is bound to a single contiguous range of memory corresponds to an identical offset within that range of memory.
 - Requested via the [VK_IMAGE_CREATE_SPARSE_BINDING_BIT](#) and [VK_BUFFER_CREATE_SPARSE_BINDING_BIT](#) bits.
 - A sparse image created using [VK_IMAGE_CREATE_SPARSE_BINDING_BIT](#) (but not [VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT](#)) supports all formats that non-sparse usage supports, and supports both [VK_IMAGE_TILING_OPTIMAL](#) and [VK_IMAGE_TILING_LINEAR](#) tiling.
- *Sparse Residency* builds on (and requires) the [sparseBinding](#) feature. It includes the following

capabilities:

- Resources do not have to be completely bound to memory before use on the device.
- Images have a prescribed sparse image block layout, allowing specific rectangular regions of the image to be bound to specific offsets in memory allocations.
- Consistency of access to unbound regions of the resource is defined by the absence or presence of `VkPhysicalDeviceSparseProperties::residencyNonResidentStrict`. If this property is present, accesses to unbound regions of the resource are well defined and behave as if the data bound is populated with all zeros; writes are discarded. When this property is absent, accesses are considered safe, but reads will return undefined values.
- Requested via the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` and `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` bits.
- Sparse residency support is advertised on a finer grain via the following features:
 - The `sparseResidencyBuffer` feature provides support for creating `VkBuffer` objects with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT`.
 - The `sparseResidencyImage2D` feature provides support for creating 2D single-sampled `VkImage` objects with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
 - The `sparseResidencyImage3D` feature provides support for creating 3D `VkImage` objects with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
 - The `sparseResidency2Samples` feature provides support for creating 2D `VkImage` objects with 2 samples and `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
 - The `sparseResidency4Samples` feature provides support for creating 2D `VkImage` objects with 4 samples and `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
 - The `sparseResidency8Samples` feature provides support for creating 2D `VkImage` objects with 8 samples and `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
 - The `sparseResidency16Samples` feature provides support for creating 2D `VkImage` objects with 16 samples and `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.

Implementations supporting `sparseResidencyImage2D` are only **required** to support sparse 2D, single-sampled images. Support for sparse 3D and MSAA images is **optional** and **can** be enabled via `sparseResidencyImage3D`, `sparseResidency2Samples`, `sparseResidency4Samples`, `sparseResidency8Samples`, and `sparseResidency16Samples`.

- A sparse image created using `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` supports all non-compressed color formats with power-of-two element size that non-sparse usage supports. Additional formats **may** also be supported and **can** be queried via `vkGetPhysicalDeviceSparseImageFormatProperties`. `VK_IMAGE_TILING_LINEAR` tiling is not supported.
- The `sparseResidencyAliased` feature provides the following capability that **can** be enabled per resource:

Allows physical memory ranges to be shared between multiple locations in the same sparse resource or between multiple sparse resources, with each binding of a memory location observing a consistent interpretation of the memory contents.

30.2. Sparse Resource API

The APIs related to sparse resources are grouped into the following categories:

- [Physical Device Features](#)
- [Physical Device Sparse Properties](#)

30.2.1. Physical Device Features

Some sparse-resource related features are reported and enabled in `VkPhysicalDeviceFeatures`. These features **must** be supported and enabled on the `VkDevice` object before applications **can** use them. See [Physical Device Features](#) for information on how to get and set enabled device features, and for more detailed explanations of these features.

Sparse Physical Device Features

- `sparseBinding`: Support for creating `VkBuffer` and `VkImage` objects with the `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` and `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` flags, respectively.
- `sparseResidencyBuffer`: Support for creating `VkBuffer` objects with the `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` flag.
- `sparseResidencyImage2D`: Support for creating 2D single-sampled `VkImage` objects with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
- `sparseResidencyImage3D`: Support for creating 3D `VkImage` objects with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
- `sparseResidency2Samples`: Support for creating 2D `VkImage` objects with 2 samples and `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
- `sparseResidency4Samples`: Support for creating 2D `VkImage` objects with 4 samples and `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
- `sparseResidency8Samples`: Support for creating 2D `VkImage` objects with 8 samples and `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
- `sparseResidency16Samples`: Support for creating 2D `VkImage` objects with 16 samples and `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`.
- `sparseResidencyAliased`: Support for creating `VkBuffer` and `VkImage` objects with the `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT` and `VK_IMAGE_CREATE_SPARSE_ALIASED_BIT` flags, respectively.

30.2.2. Physical Device Sparse Properties

Some features of the implementation are not possible to disable, and are reported to allow applications to alter their sparse resource usage accordingly. These read-only capabilities are reported in the `VkPhysicalDeviceProperties::sparseProperties` member, which is a `VkPhysicalDeviceSparseProperties` structure.

The `VkPhysicalDeviceSparseProperties` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPhysicalDeviceSparseProperties {
    VkBool32    residencyStandard2DBlockShape;
    VkBool32    residencyStandard2DMultisampleBlockShape;
    VkBool32    residencyStandard3DBlockShape;
    VkBool32    residencyAlignedMipSize;
    VkBool32    residencyNonResidentStrict;
} VkPhysicalDeviceSparseProperties;
```

- `residencyStandard2DBlockShape` **must** be `VK_FALSE` in Vulkan SC [\[SCID-8\]](#).
- `residencyStandard2DMultisampleBlockShape` **must** be `VK_FALSE` in Vulkan SC [\[SCID-8\]](#).
- `residencyStandard3DBlockShape` **must** be `VK_FALSE` in Vulkan SC [\[SCID-8\]](#).
- `residencyAlignedMipSize` **must** be `VK_FALSE` in Vulkan SC [\[SCID-8\]](#).
- `residencyNonResidentStrict` **must** be `VK_FALSE` in Vulkan SC [\[SCID-8\]](#).

Chapter 31. Window System Integration (WSI)

This chapter discusses the window system integration (WSI) between the Vulkan API and the various forms of displaying the results of rendering to a user. Since the Vulkan API **can** be used without displaying results, WSI is provided through the use of optional Vulkan extensions. This chapter provides an overview of WSI. See the appendix for additional details of each WSI extension, including which extensions **must** be enabled in order to use each of the functions described in this chapter.

31.1. WSI Platform

A platform is an abstraction for a window system, OS, etc. Some examples include MS Windows, Android, and Wayland. The Vulkan API **may** be integrated in a unique manner for each platform.

The Vulkan API does not define any type of platform object. Platform-specific WSI extensions are defined, each containing platform-specific functions for using WSI. Use of these extensions is guarded by preprocessor symbols as defined in the [Window System-Specific Header Control](#) appendix.

In order for an application to be compiled to use WSI with a given platform, it **must** either:

- `#define` the appropriate preprocessor symbol prior to including the `vulkan_sc.h` header file, or
- include `vulkan_sc_core.h` and any native platform headers, followed by the appropriate platform-specific header.

The preprocessor symbols and platform-specific headers are defined in the [Window System Extensions and Headers](#) table.

Each platform-specific extension is an instance extension. The application **must** enable instance extensions with `vkCreateInstance` before using them.

31.2. WSI Surface

Native platform surface or window objects are abstracted by surface objects, which are represented by `VkSurfaceKHR` handles:

```
// Provided by VK_KHR_surface
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSurfaceKHR)
```

The `VK_KHR_surface` extension declares the `VkSurfaceKHR` object, and provides a function for destroying `VkSurfaceKHR` objects. Separate platform-specific extensions each provide a function for creating a `VkSurfaceKHR` object for the respective platform. From the application's perspective this is an opaque handle, just like the handles of other Vulkan objects.

NOTE

On certain platforms, the Vulkan loader and ICDs **may** have conventions that treat

the handle as a pointer to a structure containing the platform-specific information about the surface. This will be described in the documentation for the loader-ICD interface, and in the `vk_icd.h` header file of the LoaderAndTools source-code repository. This does not affect the loader-layer interface; layers **may** wrap `VkSurfaceKHR` objects.

31.2.1. Platform-Independent Information

Once created, `VkSurfaceKHR` objects **can** be used in this and other extensions, in particular the `VK_KHR_swapchain` extension.

Several WSI functions return `VK_ERROR_SURFACE_LOST_KHR` if the surface becomes no longer available. After such an error, the surface (and any child swapchain, if one exists) **should** be destroyed, as there is no way to restore them to a not-lost state. Applications **may** attempt to create a new `VkSurfaceKHR` using the same native platform window object, but whether such re-creation will succeed is platform-dependent and **may** depend on the reason the surface became unavailable. A lost surface does not otherwise cause devices to be [lost](#).

To destroy a `VkSurfaceKHR` object, call:

```
// Provided by VK_KHR_surface
void vkDestroySurfaceKHR(
    VkInstance          instance,
    VkSurfaceKHR        surface,
    const VkAllocationCallbacks* pAllocator);
```

- `instance` is the instance used to create the surface.
- `surface` is the surface to destroy.
- `pAllocator` is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see [Memory Allocation](#)).

Destroying a `VkSurfaceKHR` merely severs the connection between Vulkan and the native surface, and does not imply destroying the native surface, closing a window, or similar behavior.

Valid Usage

Valid Usage (Implicit)

- VUID-vkDestroySurfaceKHR-instance-parameter
`instance` **must** be a valid `VkInstance` handle
- VUID-vkDestroySurfaceKHR-surface-parameter
If `surface` is not `VK_NULL_HANDLE`, `surface` **must** be a valid `VkSurfaceKHR` handle
- VUID-vkDestroySurfaceKHR-pAllocator-null
`pAllocator` **must** be `NULL`

- VUID-vkDestroySurfaceKHR-surface-parent

If **surface** is a valid handle, it **must** have been created, allocated, or retrieved from **instance**

Host Synchronization

- Host access to **surface** **must** be externally synchronized

31.3. Presenting Directly to Display Devices

In some environments applications **can** also present Vulkan rendering directly to display devices without using an intermediate windowing system. This **can** be useful for embedded applications, or implementing the rendering/presentation backend of a windowing system using Vulkan. The **VK_KHR_display** extension provides the functionality necessary to enumerate display devices and create **VkSurfaceKHR** objects that target displays.

31.3.1. Display Enumeration

Displays are represented by **VkDisplayKHR** handles:

```
// Provided by VK_KHR_display
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDisplayKHR)
```

Various functions are provided for enumerating the available display devices present on a Vulkan physical device. To query information about the available displays, call:

```
// Provided by VK_KHR_display
VkResult vkGetPhysicalDeviceDisplayPropertiesKHR(
    VkPhysicalDevice          physicalDevice,
    uint32_t*                 pPropertyCount,
    VkDisplayPropertiesKHR*    pProperties);
```

- **physicalDevice** is a physical device.
- **pPropertyCount** is a pointer to an integer related to the number of display devices available or queried, as described below.
- **pProperties** is either **NULL** or a pointer to an array of **VkDisplayPropertiesKHR** structures.

If **pProperties** is **NULL**, then the number of display devices available for **physicalDevice** is returned in **pPropertyCount**. Otherwise, **pPropertyCount** **must** point to a variable set by the application to the number of elements in the **pProperties** array, and on return the variable is overwritten with the number of structures actually written to **pProperties**. If the value of **pPropertyCount** is less than the number of display devices for **physicalDevice**, at most **pPropertyCount** structures will be written, and **VK_INCOMPLETE** will be returned instead of **VK_SUCCESS**, to indicate that not all the available properties were returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDeviceDisplayPropertiesKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- `VUID-vkGetPhysicalDeviceDisplayPropertiesKHR-physicalDevice-parameter` `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- `VUID-vkGetPhysicalDeviceDisplayPropertiesKHR-pPropertyCount-parameter` `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- `VUID-vkGetPhysicalDeviceDisplayPropertiesKHR-pProperties-parameter`
If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` `VkDisplayPropertiesKHR` structures

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkDisplayPropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplayPropertiesKHR {
    VkDisplayKHR          display;
    const char*           displayName;
    VkExtent2D            physicalDimensions;
    VkExtent2D            physicalResolution;
    VkSurfaceTransformFlagsKHR supportedTransforms;
    VkBool32              planeReorderPossible;
    VkBool32              persistentContent;
} VkDisplayPropertiesKHR;
```

- `display` is a handle that is used to refer to the display described here. This handle will be valid for the lifetime of the Vulkan instance.
- `displayName` is `NULL` or a pointer to a null-terminated UTF-8 string containing the name of the display. Generally, this will be the name provided by the display's EDID. If `NULL`, no suitable name is available. If not `NULL`, the string pointed to **must** remain accessible and unmodified as

long as `display` is valid.

- `physicalDimensions` describes the physical width and height of the visible portion of the display, in millimeters.
- `physicalResolution` describes the physical, native, or preferred resolution of the display.

NOTE

For devices which have no natural value to return here, implementations **should** return the maximum resolution supported.

- `supportedTransforms` is a bitmask of `VkSurfaceTransformFlagBitsKHR` describing which transforms are supported by this display.
- `planeReorderPossible` tells whether the planes on this display **can** have their z order changed. If this is `VK_TRUE`, the application **can** re-arrange the planes on this display in any order relative to each other.
- `persistentContent` tells whether the display supports self-refresh/internal buffering. If this is true, the application **can** submit persistent present operations on swapchains created against this display.

NOTE

Persistent presents **may** have higher latency, and **may** use less power when the screen content is updated infrequently, or when only a portion of the screen needs to be updated in most frames.

To query information about the available displays, call:

```
// Provided by VK_KHR_get_display_properties2
VkResult vkGetPhysicalDeviceDisplayProperties2KHR(
    VkPhysicalDevice          physicalDevice,
    uint32_t*                 pPropertyCount,
    VkDisplayProperties2KHR*   pProperties);
```

- `physicalDevice` is a physical device.
- `pPropertyCount` is a pointer to an integer related to the number of display devices available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of `VkDisplayProperties2KHR` structures.

`vkGetPhysicalDeviceDisplayProperties2KHR` behaves similarly to `vkGetPhysicalDeviceDisplayPropertiesKHR`, with the ability to return extended information via chained output structures.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDeviceDisplayProperties2KHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceDisplayProperties2KHR-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle

- VUID-vkGetPhysicalDeviceDisplayProperties2KHR-pPropertyCount-parameter
`pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceDisplayProperties2KHR-pProperties-parameter
If the value referenced by `pPropertyCount` is not 0, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` `VkDisplayProperties2KHR` structures

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkDisplayProperties2KHR` structure is defined as:

```
// Provided by VK_KHR_get_display_properties2
typedef struct VkDisplayProperties2KHR {
    VkStructureType    sType;
    void*              pNext;
    VkDisplayPropertiesKHR displayProperties;
} VkDisplayProperties2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `displayProperties` is a `VkDisplayPropertiesKHR` structure.

Valid Usage (Implicit)

- VUID-VkDisplayProperties2KHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_PROPERTIES_2_KHR`
- VUID-VkDisplayProperties2KHR-pNext-pNext
`pNext` **must** be `NULL`

Acquiring and Releasing Displays

On some platforms, access to displays is limited to a single process or native driver instance. On such platforms, some or all of the displays may not be available to Vulkan if they are already in use

by a native windowing system or other application.

To acquire permission to directly access a display in Vulkan on Windows 10, call:

```
// Provided by VK_NV_acquire_winrt_display
VkResult vkAcquireWinrtDisplayNV(
    VkPhysicalDevice      physicalDevice,
    VkDisplayKHR          display);
```

- **physicalDevice** The physical device the display is on.
- **display** The display the caller wishes to control in Vulkan.

All permissions necessary to control the display are granted to the Vulkan instance associated with **physicalDevice** until the display is released or the application is terminated. Permission to access the display **may** be revoked by events that cause Windows 10 itself to lose access to **display**. If this has happened, operations which require access to the display **must** fail with an appropriate error code. If permission to access **display** has already been acquired by another entity, the call **must** return the error code **VK_ERROR_INITIALIZATION_FAILED**.

NOTE	The Vulkan instance acquires control of a “winrt::Windows::Devices::Display::Core::DisplayTarget” by performing an operation equivalent to “winrt::Windows::Devices::Display::Core::DisplayManager.TryAcquireTarget()” on the “DisplayTarget”.

NOTE	One example of when Windows 10 loses access to a display is when the display is hot-unplugged.

NOTE	One example of when a display has already been acquired by another entity is when the Windows desktop compositor (DWM) is in control of the display. Beginning with Windows 10 version 2004 it is possible to cause DWM to release a display by using the “Advanced display settings” sub-page of the “Display settings” control panel. vkAcquireWinrtDisplayNV does not itself cause DWM to release a display; this action must be performed outside of Vulkan.

If **VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations** is **VK_TRUE**, **vkAcquireWinrtDisplayNV** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage (Implicit)

- VUID-vkAcquireWinrtDisplayNV-physicalDevice-parameter **physicalDevice** **must** be a valid **VkPhysicalDevice** handle
- VUID-vkAcquireWinrtDisplayNV-display-parameter **display** **must** be a valid **VkDisplayKHR** handle
- VUID-vkAcquireWinrtDisplayNV-display-parent

`display` **must** have been created, allocated, or retrieved from `physicalDevice`

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

When acquiring displays on Windows 10, an application may also wish to enumerate and identify them using a native handle rather than a `VkDisplayKHR` handle.

To determine the `VkDisplayKHR` handle corresponding to a “`wintr::Windows::Devices::Display::Core::DisplayTarget`”, call:

```
// Provided by VK_NV_acquire_wintr_display
VkResult vkGetWintrDisplayNV(
    VkPhysicalDevice      physicalDevice,
    uint32_t              deviceRelativeId,
    VkDisplayKHR*         pDisplay);
```

- `physicalDevice` The physical device on which to query the display handle.
- `deviceRelativeId` The value of the “`AdapterRelativeId`” property of a “`DisplayTarget`” that is enumerated by a “`DisplayAdapter`” with an “`Id`” property matching the `deviceLUID` property of a `VkPhysicalDeviceIDProperties` for `physicalDevice`.
- `pDisplay` The corresponding `VkDisplayKHR` handle will be returned here.

If there is no `VkDisplayKHR` corresponding to `deviceRelativeId` on `physicalDevice`, `VK_NULL_HANDLE` **must** be returned in `pDisplay`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetWintrDisplayNV` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetWintrDisplayNV-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetWintrDisplayNV-pDisplay-parameter

`pDisplay` **must** be a valid pointer to a `VkDisplayKHR` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To release a previously acquired display, call:

```
// Provided by VK_EXT_direct_mode_display
VkResult vkReleaseDisplayEXT(
    VkPhysicalDevice      physicalDevice,
    VkDisplayKHR          display);
```

- `physicalDevice` The physical device the display is on.
- `display` The display to release control of.

Valid Usage (Implicit)

- VUID-vkReleaseDisplayEXT-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkReleaseDisplayEXT-display-parameter `display` **must** be a valid `VkDisplayKHR` handle
- VUID-vkReleaseDisplayEXT-display-parent `display` **must** have been created, allocated, or retrieved from `physicalDevice`

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

Display Planes

Images are presented to individual planes on a display. Devices **must** support at least one plane on each display. Planes **can** be stacked and blended to composite multiple images on one display. Devices **may** support only a fixed stacking order and fixed mapping between planes and displays, or they **may** allow arbitrary application-specified stacking orders and mappings between planes and displays. To query the properties of device display planes, call:

```
// Provided by VK_KHR_display
VkResult vkGetPhysicalDeviceDisplayPlanePropertiesKHR(
    VkPhysicalDevice          physicalDevice,
    uint32_t*                 pPropertyCount,
    VkDisplayPlanePropertiesKHR* pProperties);
```

- `physicalDevice` is a physical device.
- `pPropertyCount` is a pointer to an integer related to the number of display planes available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of `VkDisplayPlanePropertiesKHR` structures.

If `pProperties` is `NULL`, then the number of display planes available for `physicalDevice` is returned in `pPropertyCount`. Otherwise, `pPropertyCount` **must** point to a variable set by the application to the number of elements in the `pProperties` array, and on return the variable is overwritten with the number of structures actually written to `pProperties`. If the value of `pPropertyCount` is less than the number of display planes for `physicalDevice`, at most `pPropertyCount` structures will be written.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDeviceDisplayPlanePropertiesKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceDisplayPlanePropertiesKHR-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceDisplayPlanePropertiesKHR-pPropertyCount-parameter `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceDisplayPlanePropertiesKHR-pProperties-parameter
If the value referenced by `pPropertyCount` is not 0, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` `VkDisplayPlanePropertiesKHR` structures

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkDisplayPlanePropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplayPlanePropertiesKHR {
    VkDisplayKHR    currentDisplay;
    uint32_t        currentStackIndex;
} VkDisplayPlanePropertiesKHR;
```

- `currentDisplay` is the handle of the display the plane is currently associated with. If the plane is not currently attached to any displays, this will be `VK_NULL_HANDLE`.
- `currentStackIndex` is the current z-order of the plane. This will be between 0 and the value returned by `vkGetPhysicalDeviceDisplayPlanePropertiesKHR` in `pPropertyCount`.

To query the properties of a device's display planes, call:

```
// Provided by VK_KHR_get_display_properties2
VkResult vkGetPhysicalDeviceDisplayPlaneProperties2KHR(
    VkPhysicalDevice    physicalDevice,
    uint32_t*           pPropertyCount,
    VkDisplayPlaneProperties2KHR* pProperties);
```

- `physicalDevice` is a physical device.
- `pPropertyCount` is a pointer to an integer related to the number of display planes available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of `VkDisplayPlaneProperties2KHR` structures.

`vkGetPhysicalDeviceDisplayPlaneProperties2KHR` behaves similarly to `vkGetPhysicalDeviceDisplayPlanePropertiesKHR`, with the ability to return extended information via chained output structures.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDeviceDisplayPlaneProperties2KHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceDisplayPlaneProperties2KHR-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle

- VUID-vkGetPhysicalDeviceDisplayPlaneProperties2KHR-pPropertyCount-parameter **pPropertyCount** **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceDisplayPlaneProperties2KHR-pProperties-parameter
If the value referenced by **pPropertyCount** is not 0, and **pProperties** is not `NULL`, **pProperties** **must** be a valid pointer to an array of **pPropertyCount** `VkDisplayPlaneProperties2KHR` structures

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkDisplayPlaneProperties2KHR` structure is defined as:

```
// Provided by VK_KHR_get_display_properties2
typedef struct VkDisplayPlaneProperties2KHR {
    VkStructureType    sType;
    void*              pNext;
    VkDisplayPlanePropertiesKHR displayPlaneProperties;
} VkDisplayPlaneProperties2KHR;
```

- **sType** is a `VkStructureType` value identifying this structure.
- **pNext** is `NULL` or a pointer to a structure extending this structure.
- **displayPlaneProperties** is a `VkDisplayPlanePropertiesKHR` structure.

Valid Usage (Implicit)

- VUID-VkDisplayPlaneProperties2KHR-sType-sType
sType **must** be `VK_STRUCTURE_TYPE_DISPLAY_PLANE_PROPERTIES_2_KHR`
- VUID-VkDisplayPlaneProperties2KHR-pNext-pNext
pNext **must** be `NULL`

To determine which displays a plane is usable with, call

```
// Provided by VK_KHR_display
VkResult vkGetDisplayPlaneSupportedDisplaysKHR(
    VkPhysicalDevice          physicalDevice,
    uint32_t                  planeIndex,
    uint32_t*                  pDisplayCount,
    VkDisplayKHR*              pDisplays);
```

- **physicalDevice** is a physical device.
- **planeIndex** is the plane which the application wishes to use, and **must** be in the range [0, physical device plane count - 1].
- **pDisplayCount** is a pointer to an integer related to the number of displays available or queried, as described below.
- **pDisplays** is either **NULL** or a pointer to an array of **VkDisplayKHR** handles.

If **pDisplays** is **NULL**, then the number of displays usable with the specified **planeIndex** for **physicalDevice** is returned in **pDisplayCount**. Otherwise, **pDisplayCount** **must** point to a variable set by the application to the number of elements in the **pDisplays** array, and on return the variable is overwritten with the number of handles actually written to **pDisplays**. If the value of **pDisplayCount** is less than the number of usable display-plane pairs for **physicalDevice**, at most **pDisplayCount** handles will be written, and **VK_INCOMPLETE** will be returned instead of **VK_SUCCESS**, to indicate that not all the available pairs were returned.

If **VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations** is **VK_TRUE**, **vkGetDisplayPlaneSupportedDisplaysKHR** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage

- VUID-vkGetDisplayPlaneSupportedDisplaysKHR-planeIndex-01249
planeIndex **must** be less than the number of display planes supported by the device as determined by calling **vkGetPhysicalDeviceDisplayPlanePropertiesKHR**

Valid Usage (Implicit)

- VUID-vkGetDisplayPlaneSupportedDisplaysKHR-physicalDevice-parameter
physicalDevice **must** be a valid **VkPhysicalDevice** handle
- VUID-vkGetDisplayPlaneSupportedDisplaysKHR-pDisplayCount-parameter
pDisplayCount **must** be a valid pointer to a **uint32_t** value
- VUID-vkGetDisplayPlaneSupportedDisplaysKHR-pDisplays-parameter
If the value referenced by **pDisplayCount** is not 0, and **pDisplays** is not **NULL**, **pDisplays** **must** be a valid pointer to an array of **pDisplayCount** **VkDisplayKHR** handles

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

Additional properties of displays are queried using specialized query functions.

Display Modes

Display modes are represented by `VkDisplayModeKHR` handles:

```
// Provided by VK_KHR_display
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDisplayModeKHR)
```

Each display has one or more supported modes associated with it by default. These built-in modes are queried by calling:

```
// Provided by VK_KHR_display
VkResult vkGetDisplayModePropertiesKHR(
    VkPhysicalDevice    physicalDevice,
    VkDisplayModeKHR    display,
    uint32_t*           pPropertyCount,
    VkDisplayModePropertiesKHR* pProperties);
```

- `physicalDevice` is the physical device associated with `display`.
- `display` is the display to query.
- `pPropertyCount` is a pointer to an integer related to the number of display modes available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of `VkDisplayModePropertiesKHR` structures.

If `pProperties` is `NULL`, then the number of display modes available on the specified `display` for `physicalDevice` is returned in `pPropertyCount`. Otherwise, `pPropertyCount` **must** point to a variable set by the application to the number of elements in the `pProperties` array, and on return the variable is overwritten with the number of structures actually written to `pProperties`. If the value of `pPropertyCount` is less than the number of display modes for `physicalDevice`, at most `pPropertyCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that

not all the available display modes were returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetDisplayModePropertiesKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetDisplayModePropertiesKHR-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetDisplayModePropertiesKHR-display-parameter `display` **must** be a valid `VkDisplayKHR` handle
- VUID-vkGetDisplayModePropertiesKHR-pPropertyCount-parameter `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetDisplayModePropertiesKHR-pProperties-parameter
If the value referenced by `pPropertyCount` is not 0, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` `VkDisplayModePropertiesKHR` structures
- VUID-vkGetDisplayModePropertiesKHR-display-parent `display` **must** have been created, allocated, or retrieved from `physicalDevice`

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkDisplayModePropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplayModePropertiesKHR {
    VkDisplayModeKHR          displayMode;
    VkDisplayModeParametersKHR parameters;
} VkDisplayModePropertiesKHR;
```

- `displayMode` is a handle to the display mode described in this structure. This handle will be valid for the lifetime of the Vulkan instance.

- `parameters` is a `VkDisplayModeParametersKHR` structure describing the display parameters associated with `displayMode`.

```
// Provided by VK_KHR_display
typedef VkFlags VkDisplayModeCreateFlagsKHR;
```

`VkDisplayModeCreateFlagsKHR` is a bitmask type for setting a mask, but is currently reserved for future use.

To query the properties of a device's built-in display modes, call:

```
// Provided by VK_KHR_get_display_properties2
VkResult vkGetDisplayModeProperties2KHR(
    VkPhysicalDevice      physicalDevice,
    VkDisplayKHR          display,
    uint32_t*             pPropertyCount,
    VkDisplayModeProperties2KHR* pProperties);
```

- `physicalDevice` is the physical device associated with `display`.
- `display` is the display to query.
- `pPropertyCount` is a pointer to an integer related to the number of display modes available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of `VkDisplayModeProperties2KHR` structures.

`vkGetDisplayModeProperties2KHR` behaves similarly to `vkGetDisplayModePropertiesKHR`, with the ability to return extended information via chained output structures.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetDisplayModeProperties2KHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetDisplayModeProperties2KHR-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetDisplayModeProperties2KHR-display-parameter `display` **must** be a valid `VkDisplayKHR` handle
- VUID-vkGetDisplayModeProperties2KHR-pPropertyCount-parameter `pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetDisplayModeProperties2KHR-pProperties-parameter
If the value referenced by `pPropertyCount` is not 0, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` `VkDisplayModeProperties2KHR` structures
- VUID-vkGetDisplayModeProperties2KHR-display-parent `display` **must** have been created, allocated, or retrieved from `physicalDevice`

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkDisplayModeProperties2KHR` structure is defined as:

```
// Provided by VK_KHR_get_display_properties2
typedef struct VkDisplayModeProperties2KHR {
    VkStructureType    sType;
    void*              pNext;
    VkDisplayModePropertiesKHR displayModeProperties;
} VkDisplayModeProperties2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `displayModeProperties` is a `VkDisplayModePropertiesKHR` structure.

Valid Usage (Implicit)

- VUID-VkDisplayModeProperties2KHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_MODE_PROPERTIES_2_KHR`
- VUID-VkDisplayModeProperties2KHR-pNext-pNext
`pNext` **must** be `NULL`

The `VkDisplayModeParametersKHR` structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplayModeParametersKHR {
    VkExtent2D    visibleRegion;
    uint32_t      refreshRate;
} VkDisplayModeParametersKHR;
```

- `visibleRegion` is the 2D extents of the visible region.
- `refreshRate` is a `uint32_t` that is the number of times the display is refreshed each second

multiplied by 1000.

NOTE For example, a 60Hz display mode would report a **refreshRate** of 60,000.

Valid Usage

- VUID-VkDisplayModeParametersKHR-width-01990
The **width** member of **visibleRegion** **must** be greater than 0
- VUID-VkDisplayModeParametersKHR-height-01991
The **height** member of **visibleRegion** **must** be greater than 0
- VUID-VkDisplayModeParametersKHR-refreshRate-01992
refreshRate **must** be greater than 0

Additional modes **may** also be created by calling:

```
// Provided by VK_KHR_display
VkResult vkCreateDisplayModeKHR(
    VkPhysicalDevice          physicalDevice,
    VkDisplayKHR              display,
    const VkDisplayModeCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator,
    VkDisplayModeKHR*         pMode);
```

- **physicalDevice** is the physical device associated with **display**.
- **display** is the display to create an additional mode for.
- **pCreateInfo** is a pointer to a **VkDisplayModeCreateInfoKHR** structure describing the new mode to create.
- **pAllocator** is the allocator used for host memory allocated for the display mode object when there is no more specific allocator available (see **Memory Allocation**).
- **pMode** is a pointer to a **VkDisplayModeKHR** handle in which the mode created is returned.

If **VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations** is **VK_TRUE**, **vkCreateDisplayModeKHR** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage (Implicit)

- VUID-vkCreateDisplayModeKHR-physicalDevice-parameter
physicalDevice **must** be a valid **VkPhysicalDevice** handle
- VUID-vkCreateDisplayModeKHR-display-parameter
display **must** be a valid **VkDisplayKHR** handle
- VUID-vkCreateDisplayModeKHR-pCreateInfo-parameter
pCreateInfo **must** be a valid pointer to a valid **VkDisplayModeCreateInfoKHR** structure

- VUID-vkCreateDisplayModeKHR-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateDisplayModeKHR-pMode-parameter
`pMode` **must** be a valid pointer to a `VkDisplayModeKHR` handle
- VUID-vkCreateDisplayModeKHR-display-parent
`display` **must** have been created, allocated, or retrieved from `physicalDevice`

Host Synchronization

- Host access to `display` **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkDisplayModeCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplayModeCreateInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkDisplayModeCreateFlagsKHR flags;
    VkDisplayModeParametersKHR parameters;
} VkDisplayModeCreateInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use, and **must** be zero.
- `parameters` is a `VkDisplayModeParametersKHR` structure describing the display parameters to use in creating the new mode. If the parameters are not compatible with the specified display, the implementation **must** return `VK_ERROR_INITIALIZATION_FAILED`.

Valid Usage (Implicit)

- VUID-VkDisplayModeCreateInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_MODE_CREATE_INFO_KHR`
- VUID-VkDisplayModeCreateInfoKHR-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkDisplayModeCreateInfoKHR-flags-zeroBitmask
`flags` **must** be `0`
- VUID-VkDisplayModeCreateInfoKHR-parameters-parameter
`parameters` **must** be a valid `VkDisplayModeParametersKHR` structure

Applications that wish to present directly to a display **must** select which layer, or “plane” of the display they wish to target, and a mode to use with the display. Each display supports at least one plane. The capabilities of a given mode and plane combination are determined by calling:

```
// Provided by VK_KHR_display
VkResult vkGetDisplayPlaneCapabilitiesKHR(
    VkPhysicalDevice          physicalDevice,
    VkDisplayModeKHR          mode,
    uint32_t                  planeIndex,
    VkDisplayPlaneCapabilitiesKHR* pCapabilities);
```

- `physicalDevice` is the physical device associated with the display specified by `mode`
- `mode` is the display mode the application intends to program when using the specified plane. Note this parameter also implicitly specifies a display.
- `planeIndex` is the plane which the application intends to use with the display, and is less than the number of display planes supported by the device.
- `pCapabilities` is a pointer to a `VkDisplayPlaneCapabilitiesKHR` structure in which the capabilities are returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetDisplayPlaneCapabilitiesKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetDisplayPlaneCapabilitiesKHR-physicalDevice-parameter
`physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetDisplayPlaneCapabilitiesKHR-mode-parameter
`mode` **must** be a valid `VkDisplayModeKHR` handle
- VUID-vkGetDisplayPlaneCapabilitiesKHR-pCapabilities-parameter
`pCapabilities` **must** be a valid pointer to a `VkDisplayPlaneCapabilitiesKHR` structure
- VUID-vkGetDisplayPlaneCapabilitiesKHR-mode-parent

mode **must** have been created, allocated, or retrieved from **physicalDevice**

Host Synchronization

- Host access to **mode** **must** be externally synchronized

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The **VkDisplayPlaneCapabilitiesKHR** structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplayPlaneCapabilitiesKHR {
    VkDisplayPlaneAlphaFlagsKHR    supportedAlpha;
    VkOffset2D                     minSrcPosition;
    VkOffset2D                     maxSrcPosition;
    VkExtent2D                     minSrcExtent;
    VkExtent2D                     maxSrcExtent;
    VkOffset2D                     minDstPosition;
    VkOffset2D                     maxDstPosition;
    VkExtent2D                     minDstExtent;
    VkExtent2D                     maxDstExtent;
} VkDisplayPlaneCapabilitiesKHR;
```

- **supportedAlpha** is a bitmask of **VkDisplayPlaneAlphaFlagBitsKHR** describing the supported alpha blending modes.
- **minSrcPosition** is the minimum source rectangle offset supported by this plane using the specified mode.
- **maxSrcPosition** is the maximum source rectangle offset supported by this plane using the specified mode. The **x** and **y** components of **maxSrcPosition** **must** each be greater than or equal to the **x** and **y** components of **minSrcPosition**, respectively.
- **minSrcExtent** is the minimum source rectangle size supported by this plane using the specified mode.
- **maxSrcExtent** is the maximum source rectangle size supported by this plane using the specified mode.

mode.

- `minDstPosition`, `maxDstPosition`, `minDstExtent`, `maxDstExtent` all have similar semantics to their corresponding `*Src*` equivalents, but apply to the output region within the mode rather than the input region within the source image. Unlike the `*Src*` offsets, `minDstPosition` and `maxDstPosition` **may** contain negative values.

The minimum and maximum position and extent fields describe the implementation limits, if any, as they apply to the specified display mode and plane. Vendors **may** support displaying a subset of a swapchain's presentable images on the specified display plane. This is expressed by returning `minSrcPosition`, `maxSrcPosition`, `minSrcExtent`, and `maxSrcExtent` values that indicate a range of possible positions and sizes which **may** be used to specify the region within the presentable images that source pixels will be read from when creating a swapchain on the specified display mode and plane.

Vendors **may** also support mapping the presentable images' content to a subset or superset of the visible region in the specified display mode. This is expressed by returning `minDstPosition`, `maxDstPosition`, `minDstExtent` and `maxDstExtent` values that indicate a range of possible positions and sizes which **may** be used to describe the region within the display mode that the source pixels will be mapped to.

Other vendors **may** support only a 1-1 mapping between pixels in the presentable images and the display mode. This **may** be indicated by returning (0,0) for `minSrcPosition`, `maxSrcPosition`, `minDstPosition`, and `maxDstPosition`, and (display mode width, display mode height) for `minSrcExtent`, `maxSrcExtent`, `minDstExtent`, and `maxDstExtent`.

The value `supportedAlpha` **must** contain at least one valid `VkDisplayPlaneAlphaFlagBitsKHR` bit.

These values indicate the limits of the implementation's individual fields. Not all combinations of values within the offset and extent ranges returned in `VkDisplayPlaneCapabilitiesKHR` are guaranteed to be supported. Presentation requests specifying unsupported combinations **may** fail.

To query the capabilities of a given mode and plane combination, call:

```
// Provided by VK_KHR_get_display_properties2
VkResult vkGetDisplayPlaneCapabilities2KHR(
    VkPhysicalDevice          physicalDevice,
    const VkDisplayPlaneInfo2KHR* pDisplayPlaneInfo,
    VkDisplayPlaneCapabilities2KHR* pCapabilities);
```

- `physicalDevice` is the physical device associated with `pDisplayPlaneInfo`.
- `pDisplayPlaneInfo` is a pointer to a `VkDisplayPlaneInfo2KHR` structure describing the plane and mode.
- `pCapabilities` is a pointer to a `VkDisplayPlaneCapabilities2KHR` structure in which the capabilities are returned.

`vkGetDisplayPlaneCapabilities2KHR` behaves similarly to `vkGetDisplayPlaneCapabilitiesKHR`, with the ability to specify extended inputs via chained input structures, and to return extended information via chained output structures.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetDisplayPlaneCapabilities2KHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetDisplayPlaneCapabilities2KHR-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetDisplayPlaneCapabilities2KHR-pDisplayPlaneInfo-parameter `pDisplayPlaneInfo` **must** be a valid pointer to a valid `VkDisplayPlaneInfo2KHR` structure
- VUID-vkGetDisplayPlaneCapabilities2KHR-pCapabilities-parameter `pCapabilities` **must** be a valid pointer to a `VkDisplayPlaneCapabilities2KHR` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkDisplayPlaneInfo2KHR` structure is defined as:

```
// Provided by VK_KHR_get_display_properties2
typedef struct VkDisplayPlaneInfo2KHR {
    VkStructureType    sType;
    const void*        pNext;
    VkDisplayModeKHR    mode;
    uint32_t            planeIndex;
} VkDisplayPlaneInfo2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `mode` is the display mode the application intends to program when using the specified plane.

NOTE This parameter also implicitly specifies a display.

- `planeIndex` is the plane which the application intends to use with the display.

The members of `VkDisplayPlaneInfo2KHR` correspond to the arguments to `vkGetDisplayPlaneCapabilitiesKHR`, with `sType` and `pNext` added for extensibility.

Valid Usage (Implicit)

- VUID-VkDisplayPlaneInfo2KHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_PLANE_INFO_2_KHR`
- VUID-VkDisplayPlaneInfo2KHR-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkDisplayPlaneInfo2KHR-mode-parameter
`mode` **must** be a valid `VkDisplayModeKHR` handle

Host Synchronization

- Host access to `mode` **must** be externally synchronized

The `VkDisplayPlaneCapabilities2KHR` structure is defined as:

```
// Provided by VK_KHR_get_display_properties2
typedef struct VkDisplayPlaneCapabilities2KHR {
    VkStructureType    sType;
    void*              pNext;
    VkDisplayPlaneCapabilitiesKHR capabilities;
} VkDisplayPlaneCapabilities2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `capabilities` is a `VkDisplayPlaneCapabilitiesKHR` structure.

Valid Usage (Implicit)

- VUID-VkDisplayPlaneCapabilities2KHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_PLANE_CAPABILITIES_2_KHR`
- VUID-VkDisplayPlaneCapabilities2KHR-pNext-pNext
`pNext` **must** be `NULL`

31.3.2. Display Control

To set the power state of a display, call:

```
// Provided by VK_EXT_display_control
VkResult vkDisplayPowerControlEXT(
    VkDevice          device,
    VkDisplayKHR      display,
    const VkDisplayPowerInfoEXT* pDisplayPowerInfo);
```

- **device** is a logical device associated with **display**.
- **display** is the display whose power state is modified.
- **pDisplayPowerInfo** is a pointer to a **VkDisplayPowerInfoEXT** structure specifying the new power state of **display**.

If **VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations** is **VK_TRUE**, **vkDisplayPowerControlEXT** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage (Implicit)

- VUID-vkDisplayPowerControlEXT-device-parameter **device** **must** be a valid **VkDevice** handle
- VUID-vkDisplayPowerControlEXT-display-parameter **display** **must** be a valid **VkDisplayKHR** handle
- VUID-vkDisplayPowerControlEXT-pDisplayPowerInfo-parameter **pDisplayPowerInfo** **must** be a valid pointer to a valid **VkDisplayPowerInfoEXT** structure
- VUID-vkDisplayPowerControlEXT-commonparent
Both of **device**, and **display** **must** have been created, allocated, or retrieved from the same **VkPhysicalDevice**

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The **VkDisplayPowerInfoEXT** structure is defined as:

```
// Provided by VK_EXT_display_control
typedef struct VkDisplayPowerInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkDisplayPowerStateEXT powerState;
} VkDisplayPowerInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `powerState` is a `VkDisplayPowerStateEXT` value specifying the new power state of the display.

Valid Usage (Implicit)

- VUID-VkDisplayPowerInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_POWER_INFO_EXT`
- VUID-VkDisplayPowerInfoEXT-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkDisplayPowerInfoEXT-powerState-parameter
`powerState` **must** be a valid `VkDisplayPowerStateEXT` value

Possible values of `VkDisplayPowerInfoEXT::powerState`, specifying the new power state of a display, are:

```
// Provided by VK_EXT_display_control
typedef enum VkDisplayPowerStateEXT {
    VK_DISPLAY_POWER_STATE_OFF_EXT = 0,
    VK_DISPLAY_POWER_STATE_SUSPEND_EXT = 1,
    VK_DISPLAY_POWER_STATE_ON_EXT = 2,
} VkDisplayPowerStateEXT;
```

- `VK_DISPLAY_POWER_STATE_OFF_EXT` specifies that the display is powered down.
- `VK_DISPLAY_POWER_STATE_SUSPEND_EXT` specifies that the display is put into a low power mode, from which it **may** be able to transition back to `VK_DISPLAY_POWER_STATE_ON_EXT` more quickly than if it were in `VK_DISPLAY_POWER_STATE_OFF_EXT`. This state **may** be the same as `VK_DISPLAY_POWER_STATE_OFF_EXT`.
- `VK_DISPLAY_POWER_STATE_ON_EXT` specifies that the display is powered on.

31.3.3. Display Surfaces

A complete display configuration includes a mode, one or more display planes and any parameters describing their behavior, and parameters describing some aspects of the images associated with those planes. Display surfaces describe the configuration of a single plane within a complete display configuration. To create a `VkSurfaceKHR` object for a display plane, call:

```
// Provided by VK_KHR_display
VkResult vkCreateDisplayPlaneSurfaceKHR(
    VkInstance                                instance,
    const VkDisplaySurfaceCreateInfoKHR*      pCreateInfo,
    const VkAllocationCallbacks*             pAllocator,
    VkSurfaceKHR*                             pSurface);
```

- **instance** is the instance corresponding to the physical device the targeted display is on.
- **pCreateInfo** is a pointer to a [VkDisplaySurfaceCreateInfoKHR](#) structure specifying which mode, plane, and other parameters to use, as described below.
- **pAllocator** is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see [Memory Allocation](#)).
- **pSurface** is a pointer to a [VkSurfaceKHR](#) handle in which the created surface is returned.

Valid Usage (Implicit)

- VUID-vkCreateDisplayPlaneSurfaceKHR-instance-parameter
instance **must** be a valid [VkInstance](#) handle
- VUID-vkCreateDisplayPlaneSurfaceKHR-pCreateInfo-parameter
pCreateInfo **must** be a valid pointer to a valid [VkDisplaySurfaceCreateInfoKHR](#) structure
- VUID-vkCreateDisplayPlaneSurfaceKHR-pAllocator-null
pAllocator **must** be **NULL**
- VUID-vkCreateDisplayPlaneSurfaceKHR-pSurface-parameter
pSurface **must** be a valid pointer to a [VkSurfaceKHR](#) handle

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The [VkDisplaySurfaceCreateInfoKHR](#) structure is defined as:

```
// Provided by VK_KHR_display
typedef struct VkDisplaySurfaceCreateInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkDisplaySurfaceCreateFlagsKHR flags;
    VkDisplayModeKHR    displayMode;
    uint32_t            planeIndex;
    uint32_t            planeStackIndex;
    VkSurfaceTransformFlagBitsKHR transform;
    float               globalAlpha;
    VkDisplayPlaneAlphaFlagBitsKHR alphaMode;
    VkExtent2D          imageExtent;
} VkDisplaySurfaceCreateInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use, and **must** be zero.
- `displayMode` is a `VkDisplayModeKHR` handle specifying the mode to use when displaying this surface.
- `planeIndex` is the plane on which this surface appears.
- `planeStackIndex` is the z-order of the plane.
- `transform` is a `VkSurfaceTransformFlagBitsKHR` value specifying the transformation to apply to images as part of the scanout operation.
- `globalAlpha` is the global alpha value. This value is ignored if `alphaMode` is not `VK_DISPLAY_PLANE_ALPHA_GLOBAL_BIT_KHR`.
- `alphaMode` is a `VkDisplayPlaneAlphaFlagBitsKHR` value specifying the type of alpha blending to use.
- `imageExtent` is the size of the presentable images to use with the surface.

NOTE

Creating a display surface **must** not modify the state of the displays, planes, or other resources it names. For example, it **must** not apply the specified mode to be set on the associated display. Application of display configuration occurs as a side effect of presenting to a display surface.

Valid Usage

- VUID-VkDisplaySurfaceCreateInfoKHR-planeIndex-01252
`planeIndex` **must** be less than the number of display planes supported by the device as determined by calling `vkGetPhysicalDeviceDisplayPlanePropertiesKHR`
- VUID-VkDisplaySurfaceCreateInfoKHR-planeReorderPossible-01253
If the `planeReorderPossible` member of the `VkDisplayPropertiesKHR` structure returned by `vkGetPhysicalDeviceDisplayPropertiesKHR` for the display corresponding to `displayMode` is `VK_TRUE` then `planeStackIndex` **must** be less than the number of display planes supported

by the device as determined by calling `vkGetPhysicalDeviceDisplayPlanePropertiesKHR`; otherwise `planeStackIndex` **must** equal the `currentStackIndex` member of `VkDisplayPlanePropertiesKHR` returned by `vkGetPhysicalDeviceDisplayPlanePropertiesKHR` for the display plane corresponding to `displayMode`

- VUID-VkDisplaySurfaceCreateInfoKHR-alphaMode-01254
If `alphaMode` is `VK_DISPLAY_PLANE_ALPHA_GLOBAL_BIT_KHR` then `globalAlpha` **must** be between 0 and 1, inclusive
- VUID-VkDisplaySurfaceCreateInfoKHR-alphaMode-01255
`alphaMode` **must** be one of the bits present in the `supportedAlpha` member of `VkDisplayPlaneCapabilitiesKHR` for the display plane corresponding to `displayMode`
- VUID-VkDisplaySurfaceCreateInfoKHR-transform-06740
`transform` **must** be one of the bits present in the `supportedTransforms` member of `VkDisplayPropertiesKHR` for the display corresponding to `displayMode`
- VUID-VkDisplaySurfaceCreateInfoKHR-width-01256
The `width` and `height` members of `imageExtent` **must** be less than or equal to `VkPhysicalDeviceLimits::maxImageDimension2D`

Valid Usage (Implicit)

- VUID-VkDisplaySurfaceCreateInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_SURFACE_CREATE_INFO_KHR`
- VUID-VkDisplaySurfaceCreateInfoKHR-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkDisplaySurfaceCreateInfoKHR-flags-zeroBitmask
`flags` **must** be 0
- VUID-VkDisplaySurfaceCreateInfoKHR-displayMode-parameter
`displayMode` **must** be a valid `VkDisplayModeKHR` handle
- VUID-VkDisplaySurfaceCreateInfoKHR-transform-parameter
`transform` **must** be a valid `VkSurfaceTransformFlagBitsKHR` value
- VUID-VkDisplaySurfaceCreateInfoKHR-alphaMode-parameter
`alphaMode` **must** be a valid `VkDisplayPlaneAlphaFlagBitsKHR` value

```
// Provided by VK_KHR_display
typedef VkFlags VkDisplaySurfaceCreateFlagsKHR;
```

`VkDisplaySurfaceCreateFlagsKHR` is a bitmask type for setting a mask, but is currently reserved for future use.

Bits which **can** be set in `VkDisplaySurfaceCreateInfoKHR::alphaMode`, specifying the type of alpha blending to use on a display, are:

```
// Provided by VK_KHR_display
typedef enum VkDisplayPlaneAlphaFlagBitsKHR {
    VK_DISPLAY_PLANE_ALPHA_OPAQUE_BIT_KHR = 0x00000001,
    VK_DISPLAY_PLANE_ALPHA_GLOBAL_BIT_KHR = 0x00000002,
    VK_DISPLAY_PLANE_ALPHA_PER_PIXEL_BIT_KHR = 0x00000004,
    VK_DISPLAY_PLANE_ALPHA_PER_PIXEL_PREMULTIPLIED_BIT_KHR = 0x00000008,
} VkDisplayPlaneAlphaFlagBitsKHR;
```

- **VK_DISPLAY_PLANE_ALPHA_OPAQUE_BIT_KHR** specifies that the source image will be treated as opaque.
- **VK_DISPLAY_PLANE_ALPHA_GLOBAL_BIT_KHR** specifies that a global alpha value **must** be specified that will be applied to all pixels in the source image.
- **VK_DISPLAY_PLANE_ALPHA_PER_PIXEL_BIT_KHR** specifies that the alpha value will be determined by the alpha component of the source image's pixels. If the source format contains no alpha values, no blending will be applied. The source alpha values are not premultiplied into the source image's other color components.
- **VK_DISPLAY_PLANE_ALPHA_PER_PIXEL_PREMULTIPLIED_BIT_KHR** is equivalent to **VK_DISPLAY_PLANE_ALPHA_PER_PIXEL_BIT_KHR**, except the source alpha values are assumed to be premultiplied into the source image's other color components.

```
// Provided by VK_KHR_display
typedef VkFlags VkDisplayPlaneAlphaFlagsKHR;
```

VkDisplayPlaneAlphaFlagsKHR is a bitmask type for setting a mask of zero or more **VkDisplayPlaneAlphaFlagBitsKHR**.

31.3.4. Presenting to Headless Surfaces

Vulkan rendering can be presented to a headless surface, where the presentation operation is a no-op producing no externally-visible result.

NOTE

Because there is no real presentation target, the headless presentation engine may be extended to impose an arbitrary or customizable set of restrictions and features. This makes it a useful portable test target for applications targeting a wide range of presentation engines where the actual target presentation engines might be scarce, unavailable or otherwise undesirable or inconvenient to use for general Vulkan application development.

The usual surface query mechanisms must be used to determine the actual restrictions and features of the implementation.

To create a headless **VkSurfaceKHR** object, call:


```
// Provided by VK_EXT_headless_surface
VkResult vkCreateHeadlessSurfaceEXT(
    VkInstance                                instance,
    const VkHeadlessSurfaceCreateInfoEXT*    pCreateInfo,
    const VkAllocationCallbacks*            pAllocator,
    VkSurfaceKHR*                            pSurface);
```

- **instance** is the instance to associate the surface with.
- **pCreateInfo** is a pointer to a [VkHeadlessSurfaceCreateInfoEXT](#) structure containing parameters affecting the creation of the surface object.
- **pAllocator** is the allocator used for host memory allocated for the surface object when there is no more specific allocator available (see [Memory Allocation](#)).
- **pSurface** is a pointer to a [VkSurfaceKHR](#) handle in which the created surface object is returned.

Valid Usage (Implicit)

- VUID-vkCreateHeadlessSurfaceEXT-instance-parameter **instance** **must** be a valid [VkInstance](#) handle
- VUID-vkCreateHeadlessSurfaceEXT-pCreateInfo-parameter **pCreateInfo** **must** be a valid pointer to a valid [VkHeadlessSurfaceCreateInfoEXT](#) structure
- VUID-vkCreateHeadlessSurfaceEXT-pAllocator-null **pAllocator** **must** be **NULL**
- VUID-vkCreateHeadlessSurfaceEXT-pSurface-parameter **pSurface** **must** be a valid pointer to a [VkSurfaceKHR](#) handle

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The [VkHeadlessSurfaceCreateInfoEXT](#) structure is defined as:

```
// Provided by VK_EXT_headless_surface
typedef struct VkHeadlessSurfaceCreateInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkHeadlessSurfaceCreateFlagsEXT flags;
} VkHeadlessSurfaceCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is reserved for future use.

Valid Usage (Implicit)

- VUID-VkHeadlessSurfaceCreateInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_HEADLESS_SURFACE_CREATE_INFO_EXT`
- VUID-VkHeadlessSurfaceCreateInfoEXT-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkHeadlessSurfaceCreateInfoEXT-flags-zero bitmask
`flags` **must** be 0

For headless surfaces, `currentExtent` is the reserved value (0xFFFFFFFF, 0xFFFFFFFF). Whatever the application sets a swapchain's `imageExtent` to will be the size of the surface, after the first image is presented.

```
// Provided by VK_EXT_headless_surface
typedef VkFlags VkHeadlessSurfaceCreateFlagsEXT;
```

`VkHeadlessSurfaceCreateFlagsEXT` is a bitmask type for setting a mask, but is currently reserved for future use.

31.4. Querying for WSI Support

Not all physical devices will include WSI support. Within a physical device, not all queue families will support presentation. WSI support and compatibility **can** be determined in a platform-neutral manner (which determines support for presentation to a particular surface object) and additionally **may** be determined in platform-specific manners (which determine support for presentation on the specified physical device but do not guarantee support for presentation to a particular surface object).

To determine whether a queue family of a physical device supports presentation to a given surface, call:

```
// Provided by VK_KHR_surface
VkResult vkGetPhysicalDeviceSurfaceSupportKHR(
    VkPhysicalDevice          physicalDevice,
    uint32_t                  queueFamilyIndex,
    VkSurfaceKHR              surface,
    VkBool32*                 pSupported);
```

- `physicalDevice` is the physical device.
- `queueFamilyIndex` is the queue family.
- `surface` is the surface.
- `pSupported` is a pointer to a `VkBool32`. `VK_TRUE` indicates support, and `VK_FALSE` indicates no support.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDeviceSurfaceSupportKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetPhysicalDeviceSurfaceSupportKHR-queueFamilyIndex-01269 `queueFamilyIndex` **must** be less than `pQueueFamilyPropertyCount` returned by `vkGetPhysicalDeviceQueueFamilyProperties` for the given `physicalDevice`

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfaceSupportKHR-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceSurfaceSupportKHR-surface-parameter `surface` **must** be a valid `VkSurfaceKHR` handle
- VUID-vkGetPhysicalDeviceSurfaceSupportKHR-pSupported-parameter `pSupported` **must** be a valid pointer to a `VkBool32` value
- VUID-vkGetPhysicalDeviceSurfaceSupportKHR-commonparent Both of `physicalDevice`, and `surface` **must** have been created, allocated, or retrieved from the same `VkInstance`

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`

- `VK_ERROR_SURFACE_LOST_KHR`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

31.5. Surface Queries

The capabilities of a swapchain targeting a surface are the intersection of the capabilities of the WSI platform, the native window or display, and the physical device. The resulting capabilities **can** be obtained with the queries listed below in this section.

NOTE

In addition to the surface capabilities as obtained by surface queries below, swapchain images are also subject to ordinary image creation limits as reported by [vkGetPhysicalDeviceImageFormatProperties](#). As an application is instructed by the appropriate Valid Usage sections, both the surface capabilities and the image creation limits have to be satisfied whenever swapchain images are created.

31.5.1. Surface Capabilities

To query the basic capabilities of a surface, needed in order to create a swapchain, call:

```
// Provided by VK_KHR_surface
VkResult vkGetPhysicalDeviceSurfaceCapabilitiesKHR(
    VkPhysicalDevice      physicalDevice,
    VkSurfaceKHR          surface,
    VkSurfaceCapabilitiesKHR* pSurfaceCapabilities);
```

- `physicalDevice` is the physical device that will be associated with the swapchain to be created, as described for [vkCreateSwapchainKHR](#).
- `surface` is the surface that will be associated with the swapchain.
- `pSurfaceCapabilities` is a pointer to a [VkSurfaceCapabilitiesKHR](#) structure in which the capabilities are returned.

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is `VK_TRUE`, `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetPhysicalDeviceSurfaceCapabilitiesKHR-surface-06211
`surface` **must** be supported by `physicalDevice`, as reported by [vkGetPhysicalDeviceSurfaceSupportKHR](#) or an equivalent platform-specific mechanism

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfaceCapabilitiesKHR-physicalDevice-parameter **physicalDevice** **must** be a valid [VkPhysicalDevice](#) handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilitiesKHR-surface-parameter **surface** **must** be a valid [VkSurfaceKHR](#) handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilitiesKHR-pSurfaceCapabilities-parameter **pSurfaceCapabilities** **must** be a valid pointer to a [VkSurfaceCapabilitiesKHR](#) structure
- VUID-vkGetPhysicalDeviceSurfaceCapabilitiesKHR-commonparent
Both of **physicalDevice**, and **surface** **must** have been created, allocated, or retrieved from the same [VkInstance](#)

Return Codes

Success

- [VK_SUCCESS](#)

Failure

- [VK_ERROR_OUT_OF_DEVICE_MEMORY](#)
- [VK_ERROR_OUT_OF_HOST_MEMORY](#)
- [VK_ERROR_SURFACE_LOST_KHR](#)
- [VK_ERROR_UNKNOWN](#)
- [VK_ERROR_VALIDATION_FAILED](#)

The [VkSurfaceCapabilitiesKHR](#) structure is defined as:

```
// Provided by VK_KHR_surface
typedef struct VkSurfaceCapabilitiesKHR {
    uint32_t          minImageCount;
    uint32_t          maxImageCount;
    VkExtent2D        currentExtent;
    VkExtent2D        minImageExtent;
    VkExtent2D        maxImageExtent;
    uint32_t          maxImageArrayLayers;
    VkSurfaceTransformFlagsKHR supportedTransforms;
    VkSurfaceTransformFlagBitsKHR currentTransform;
    VkCompositeAlphaFlagsKHR supportedCompositeAlpha;
    VkImageUsageFlags supportedUsageFlags;
} VkSurfaceCapabilitiesKHR;
```

- **minImageCount** is the minimum number of images the specified device supports for a swapchain created for the surface, and will be at least one.
- **maxImageCount** is the maximum number of images the specified device supports for a swapchain

created for the surface, and will be either 0, or greater than or equal to `minImageCount`. A value of 0 means that there is no limit on the number of images, though there **may** be limits related to the total amount of memory used by presentable images.

- `currentExtent` is the current width and height of the surface, or the special value (0xFFFFFFFF, 0xFFFFFFFF) indicating that the surface size will be determined by the extent of a swapchain targeting the surface.
- `minImageExtent` contains the smallest valid swapchain extent for the surface on the specified device. The `width` and `height` of the extent will each be less than or equal to the corresponding `width` and `height` of `currentExtent`, unless `currentExtent` has the special value described above.
- `maxImageExtent` contains the largest valid swapchain extent for the surface on the specified device. The `width` and `height` of the extent will each be greater than or equal to the corresponding `width` and `height` of `minImageExtent`. The `width` and `height` of the extent will each be greater than or equal to the corresponding `width` and `height` of `currentExtent`, unless `currentExtent` has the special value described above.
- `maxImageArrayLayers` is the maximum number of layers presentable images **can** have for a swapchain created for this device and surface, and will be at least one.
- `supportedTransforms` is a bitmask of `VkSurfaceTransformFlagBitsKHR` indicating the presentation transforms supported for the surface on the specified device. At least one bit will be set.
- `currentTransform` is `VkSurfaceTransformFlagBitsKHR` value indicating the surface's current transform relative to the presentation engine's natural orientation.
- `supportedCompositeAlpha` is a bitmask of `VkCompositeAlphaFlagBitsKHR`, representing the alpha compositing modes supported by the presentation engine for the surface on the specified device, and at least one bit will be set. Opaque composition **can** be achieved in any alpha compositing mode by either using an image format that has no alpha component, or by ensuring that all pixels in the presentable images have an alpha value of 1.0.
- `supportedUsageFlags` is a bitmask of `VkImageUsageFlagBits` representing the ways the application **can** use the presentable images of a swapchain created with `VkPresentModeKHR` set to `VK_PRESENT_MODE_IMMEDIATE_KHR`, `VK_PRESENT_MODE_MAILBOX_KHR`, `VK_PRESENT_MODE_FIFO_KHR` or `VK_PRESENT_MODE_FIFO_RELAXED_KHR` for the surface on the specified device. `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` **must** be included in the set. Implementations **may** support additional usages.

NOTE	Supported usage flags of a presentable image when using <code>VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR</code> or <code>VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR</code> presentation mode are provided by <code>VkSharedPresentSurfaceCapabilitiesKHR::sharedPresentSupportedUsageFlags</code> .

NOTE	Formulas such as <code>min(N, maxImageCount)</code> are not correct, since <code>maxImageCount</code> may be zero.

To query the basic capabilities of a surface defined by the core or extensions, call:

```
// Provided by VK_KHR_get_surface_capabilities2
VkResult vkGetPhysicalDeviceSurfaceCapabilities2KHR(
    VkPhysicalDevice          physicalDevice,
    const VkPhysicalDeviceSurfaceInfo2KHR* pSurfaceInfo,
    VkSurfaceCapabilities2KHR* pSurfaceCapabilities);
```

- `physicalDevice` is the physical device that will be associated with the swapchain to be created, as described for `vkCreateSwapchainKHR`.
- `pSurfaceInfo` is a pointer to a `VkPhysicalDeviceSurfaceInfo2KHR` structure describing the surface and other fixed parameters that would be consumed by `vkCreateSwapchainKHR`.
- `pSurfaceCapabilities` is a pointer to a `VkSurfaceCapabilities2KHR` structure in which the capabilities are returned.

`vkGetPhysicalDeviceSurfaceCapabilities2KHR` behaves similarly to `vkGetPhysicalDeviceSurfaceCapabilitiesKHR`, with the ability to specify extended inputs via chained input structures, and to return extended information via chained output structures.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDeviceSurfaceCapabilities2KHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pSurfaceInfo-06521
`pSurfaceInfo->surface` **must** be a valid `VkSurfaceKHR` handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pSurfaceInfo-06522
`pSurfaceInfo->surface` **must** be supported by `physicalDevice`, as reported by `vkGetPhysicalDeviceSurfaceSupportKHR` or an equivalent platform-specific mechanism

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-physicalDevice-parameter
`physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pSurfaceInfo-parameter
`pSurfaceInfo` **must** be a valid pointer to a valid `VkPhysicalDeviceSurfaceInfo2KHR` structure
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2KHR-pSurfaceCapabilities-parameter
`pSurfaceCapabilities` **must** be a valid pointer to a `VkSurfaceCapabilities2KHR` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_SURFACE_LOST_KHR`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkPhysicalDeviceSurfaceInfo2KHR` structure is defined as:

```
// Provided by VK_KHR_get_surface_capabilities2
typedef struct VkPhysicalDeviceSurfaceInfo2KHR {
    VkStructureType    sType;
    const void*        pNext;
    VkSurfaceKHR       surface;
} VkPhysicalDeviceSurfaceInfo2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `surface` is the surface that will be associated with the swapchain.

The members of `VkPhysicalDeviceSurfaceInfo2KHR` correspond to the arguments to `vkGetPhysicalDeviceSurfaceCapabilitiesKHR`, with `sType` and `pNext` added for extensibility.

Valid Usage

- VUID-VkPhysicalDeviceSurfaceInfo2KHR-surface-07919
If `surface` is not `VK_NULL_HANDLE`, `surface` **must** be a valid `VkSurfaceKHR` handle

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSurfaceInfo2KHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SURFACE_INFO_2_KHR`
- VUID-VkPhysicalDeviceSurfaceInfo2KHR-pNext-pNext
`pNext` **must** be `NULL`

The `VkSurfaceCapabilities2KHR` structure is defined as:

```
// Provided by VK_KHR_get_surface_capabilities2
typedef struct VkSurfaceCapabilities2KHR {
    VkStructureType    sType;
    void*              pNext;
    VkSurfaceCapabilitiesKHR surfaceCapabilities;
} VkSurfaceCapabilities2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `surfaceCapabilities` is a `VkSurfaceCapabilitiesKHR` structure describing the capabilities of the specified surface.

Valid Usage (Implicit)

- VUID-VkSurfaceCapabilities2KHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_KHR`
- VUID-VkSurfaceCapabilities2KHR-pNext-pNext
`pNext` **must** be `NULL` or a pointer to a valid instance of `VkSharedPresentSurfaceCapabilitiesKHR`
- VUID-VkSurfaceCapabilities2KHR-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique

The `VkSharedPresentSurfaceCapabilitiesKHR` structure is defined as:

```
// Provided by VK_KHR_shared_presentable_image
typedef struct VkSharedPresentSurfaceCapabilitiesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkImageUsageFlags  sharedPresentSupportedUsageFlags;
} VkSharedPresentSurfaceCapabilitiesKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `sharedPresentSupportedUsageFlags` is a bitmask of `VkImageUsageFlagBits` representing the ways the application **can** use the shared presentable image from a swapchain created with `VkPresentModeKHR` set to `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` or `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR` for the surface on the specified device. `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` **must** be included in the set but implementations **may** support additional usages.

Valid Usage (Implicit)

- VUID-VkSharedPresentSurfaceCapabilitiesKHR-sType-sType

sType **must** be `VK_STRUCTURE_TYPE_SHARED_PRESENT_SURFACE_CAPABILITIES_KHR`

To query the basic capabilities of a surface, needed in order to create a swapchain, call:

```
// Provided by VK_EXT_display_surface_counter
VkResult vkGetPhysicalDeviceSurfaceCapabilities2EXT(
    VkPhysicalDevice    physicalDevice,
    VkSurfaceKHR        surface,
    VkSurfaceCapabilities2EXT* pSurfaceCapabilities);
```

- **physicalDevice** is the physical device that will be associated with the swapchain to be created, as described for `vkCreateSwapchainKHR`.
- **surface** is the surface that will be associated with the swapchain.
- **pSurfaceCapabilities** is a pointer to a `VkSurfaceCapabilities2EXT` structure in which the capabilities are returned.

`vkGetPhysicalDeviceSurfaceCapabilities2EXT` behaves similarly to `vkGetPhysicalDeviceSurfaceCapabilitiesKHR`, with the ability to return extended information by adding extending structures to the **pNext** chain of its **pSurfaceCapabilities** parameter.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDeviceSurfaceCapabilities2EXT` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetPhysicalDeviceSurfaceCapabilities2EXT-surface-06211
surface **must** be supported by **physicalDevice**, as reported by `vkGetPhysicalDeviceSurfaceSupportKHR` or an equivalent platform-specific mechanism

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfaceCapabilities2EXT-physicalDevice-parameter
physicalDevice **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2EXT-surface-parameter
surface **must** be a valid `VkSurfaceKHR` handle
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2EXT-pSurfaceCapabilities-parameter
pSurfaceCapabilities **must** be a valid pointer to a `VkSurfaceCapabilities2EXT` structure
- VUID-vkGetPhysicalDeviceSurfaceCapabilities2EXT-commonparent
Both of **physicalDevice**, and **surface** **must** have been created, allocated, or retrieved from the same `VkInstance`

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_SURFACE_LOST_KHR`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkSurfaceCapabilities2EXT` structure is defined as:

```
// Provided by VK_EXT_display_surface_counter
typedef struct VkSurfaceCapabilities2EXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           minImageCount;
    uint32_t           maxImageCount;
    VkExtent2D         currentExtent;
    VkExtent2D         minImageExtent;
    VkExtent2D         maxImageExtent;
    uint32_t           maxImageArrayLayers;
    VkSurfaceTransformFlagsKHR supportedTransforms;
    VkSurfaceTransformFlagBitsKHR currentTransform;
    VkCompositeAlphaFlagsKHR supportedCompositeAlpha;
    VkImageUsageFlags  supportedUsageFlags;
    VkSurfaceCounterFlagsEXT supportedSurfaceCounters;
} VkSurfaceCapabilities2EXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `minImageCount` is the minimum number of images the specified device supports for a swapchain created for the surface, and will be at least one.
- `maxImageCount` is the maximum number of images the specified device supports for a swapchain created for the surface, and will be either 0, or greater than or equal to `minImageCount`. A value of 0 means that there is no limit on the number of images, though there **may** be limits related to the total amount of memory used by presentable images.
- `currentExtent` is the current width and height of the surface, or the special value (0xFFFFFFFF, 0xFFFFFFFF) indicating that the surface size will be determined by the extent of a swapchain targeting the surface.
- `minImageExtent` contains the smallest valid swapchain extent for the surface on the specified

device. The **width** and **height** of the extent will each be less than or equal to the corresponding **width** and **height** of **currentExtent**, unless **currentExtent** has the special value described above.

- **maxImageExtent** contains the largest valid swapchain extent for the surface on the specified device. The **width** and **height** of the extent will each be greater than or equal to the corresponding **width** and **height** of **minImageExtent**. The **width** and **height** of the extent will each be greater than or equal to the corresponding **width** and **height** of **currentExtent**, unless **currentExtent** has the special value described above.
- **maxImageArrayLayers** is the maximum number of layers presentable images **can** have for a swapchain created for this device and surface, and will be at least one.
- **supportedTransforms** is a bitmask of **VkSurfaceTransformFlagBitsKHR** indicating the presentation transforms supported for the surface on the specified device. At least one bit will be set.
- **currentTransform** is **VkSurfaceTransformFlagBitsKHR** value indicating the surface's current transform relative to the presentation engine's natural orientation.
- **supportedCompositeAlpha** is a bitmask of **VkCompositeAlphaFlagBitsKHR**, representing the alpha compositing modes supported by the presentation engine for the surface on the specified device, and at least one bit will be set. Opaque composition **can** be achieved in any alpha compositing mode by either using an image format that has no alpha component, or by ensuring that all pixels in the presentable images have an alpha value of 1.0.
- **supportedUsageFlags** is a bitmask of **VkImageUsageFlagBits** representing the ways the application **can** use the presentable images of a swapchain created with **VkPresentModeKHR** set to **VK_PRESENT_MODE_IMMEDIATE_KHR**, **VK_PRESENT_MODE_MAILBOX_KHR**, **VK_PRESENT_MODE_FIFO_KHR** or **VK_PRESENT_MODE_FIFO_RELAXED_KHR** for the surface on the specified device. **VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT** **must** be included in the set. Implementations **may** support additional usages.
- **supportedSurfaceCounters** is a bitmask of **VkSurfaceCounterFlagBitsEXT** indicating the supported surface counter types.

Valid Usage

- VUID-VkSurfaceCapabilities2EXT-supportedSurfaceCounters-01246
supportedSurfaceCounters **must** not include **VK_SURFACE_COUNTER_VBLANK_BIT_EXT** unless the surface queried is a **display surface**

Valid Usage (Implicit)

- VUID-VkSurfaceCapabilities2EXT-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_EXT**
- VUID-VkSurfaceCapabilities2EXT-pNext-pNext
pNext **must** be **NULL**

Bits which **can** be set in **VkSurfaceCapabilities2EXT::supportedSurfaceCounters**, indicating supported surface counter types, are:

```
// Provided by VK_EXT_display_surface_counter
typedef enum VkSurfaceCounterFlagBitsEXT {
    VK_SURFACE_COUNTER_VBLANK_BIT_EXT = 0x00000001,
    // VK_SURFACE_COUNTER_VBLANK_EXT is a legacy alias
    VK_SURFACE_COUNTER_VBLANK_EXT = VK_SURFACE_COUNTER_VBLANK_BIT_EXT,
} VkSurfaceCounterFlagBitsEXT;
```

- **VK_SURFACE_COUNTER_VBLANK_BIT_EXT** specifies a counter incrementing once every time a vertical blanking period occurs on the display associated with the surface.

```
// Provided by VK_EXT_display_surface_counter
typedef VkFlags VkSurfaceCounterFlagsEXT;
```

VkSurfaceCounterFlagsEXT is a bitmask type for setting a mask of zero or more **VkSurfaceCounterFlagBitsEXT**.

Bits which **may** be set in **VkSurfaceCapabilitiesKHR::supportedTransforms** indicating the presentation transforms supported for the surface on the specified device, and possible values of **VkSurfaceCapabilitiesKHR::currentTransform** indicating the surface's current transform relative to the presentation engine's natural orientation, are:

```
// Provided by VK_KHR_surface
typedef enum VkSurfaceTransformFlagBitsKHR {
    VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR = 0x00000001,
    VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR = 0x00000002,
    VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR = 0x00000004,
    VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR = 0x00000008,
    VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_BIT_KHR = 0x00000010,
    VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_90_BIT_KHR = 0x00000020,
    VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_180_BIT_KHR = 0x00000040,
    VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_270_BIT_KHR = 0x00000080,
    VK_SURFACE_TRANSFORM_INHERIT_BIT_KHR = 0x00000100,
} VkSurfaceTransformFlagBitsKHR;
```

- **VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR** specifies that image content is presented without being transformed.
- **VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR** specifies that image content is rotated 90 degrees clockwise.
- **VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR** specifies that image content is rotated 180 degrees clockwise.
- **VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR** specifies that image content is rotated 270 degrees clockwise.
- **VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_BIT_KHR** specifies that image content is mirrored horizontally.

- `VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_90_BIT_KHR` specifies that image content is mirrored horizontally, then rotated 90 degrees clockwise.
- `VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_180_BIT_KHR` specifies that image content is mirrored horizontally, then rotated 180 degrees clockwise.
- `VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_270_BIT_KHR` specifies that image content is mirrored horizontally, then rotated 270 degrees clockwise.
- `VK_SURFACE_TRANSFORM_INHERIT_BIT_KHR` specifies that the presentation transform is not specified, and is instead determined by platform-specific considerations and mechanisms outside Vulkan.

```
// Provided by VK_KHR_display
typedef VkFlags VkSurfaceTransformFlagsKHR;
```

`VkSurfaceTransformFlagsKHR` is a bitmask type for setting a mask of zero or more `VkSurfaceTransformFlagBitsKHR`.

The `supportedCompositeAlpha` member is of type `VkCompositeAlphaFlagBitsKHR`, containing the following values:

```
// Provided by VK_KHR_surface
typedef enum VkCompositeAlphaFlagBitsKHR {
    VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR = 0x00000001,
    VK_COMPOSITE_ALPHA_PRE_MULTIPLIED_BIT_KHR = 0x00000002,
    VK_COMPOSITE_ALPHA_POST_MULTIPLIED_BIT_KHR = 0x00000004,
    VK_COMPOSITE_ALPHA_INHERIT_BIT_KHR = 0x00000008,
} VkCompositeAlphaFlagBitsKHR;
```

These values are described as follows:

- `VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR`: The alpha component, if it exists, of the images is ignored in the compositing process. Instead, the image is treated as if it has a constant alpha of 1.0.
- `VK_COMPOSITE_ALPHA_PRE_MULTIPLIED_BIT_KHR`: The alpha component, if it exists, of the images is respected in the compositing process. The non-alpha components of the image are expected to already be multiplied by the alpha component by the application.
- `VK_COMPOSITE_ALPHA_POST_MULTIPLIED_BIT_KHR`: The alpha component, if it exists, of the images is respected in the compositing process. The non-alpha components of the image are not expected to already be multiplied by the alpha component by the application; instead, the compositor will multiply the non-alpha components of the image by the alpha component during compositing.
- `VK_COMPOSITE_ALPHA_INHERIT_BIT_KHR`: The way in which the presentation engine treats the alpha component in the images is unknown to the Vulkan API. Instead, the application is responsible for setting the composite alpha blending mode using native window system commands. If the application does not set the blending mode using native window system commands, then a platform-specific default will be used.


```
// Provided by VK_KHR_surface
typedef VkFlags VkCompositeAlphaFlagsKHR;
```

`VkCompositeAlphaFlagsKHR` is a bitmask type for setting a mask of zero or more `VkCompositeAlphaFlagBitsKHR`.

31.5.2. Surface Format Support

To query the supported swapchain format-color space pairs for a surface, call:

```
// Provided by VK_KHR_surface
VkResult vkGetPhysicalDeviceSurfaceFormatsKHR(
    VkPhysicalDevice          physicalDevice,
    VkSurfaceKHR              surface,
    uint32_t*                 pSurfaceFormatCount,
    VkSurfaceFormatKHR*       pSurfaceFormats);
```

- `physicalDevice` is the physical device that will be associated with the swapchain to be created, as described for `vkCreateSwapchainKHR`.
- `surface` is the surface that will be associated with the swapchain.
- `pSurfaceFormatCount` is a pointer to an integer related to the number of format pairs available or queried, as described below.
- `pSurfaceFormats` is either `NULL` or a pointer to an array of `VkSurfaceFormatKHR` structures.

If `pSurfaceFormats` is `NULL`, then the number of format pairs supported for the given `surface` is returned in `pSurfaceFormatCount`. Otherwise, `pSurfaceFormatCount` **must** point to a variable set by the application to the number of elements in the `pSurfaceFormats` array, and on return the variable is overwritten with the number of structures actually written to `pSurfaceFormats`. If the value of `pSurfaceFormatCount` is less than the number of format pairs supported, at most `pSurfaceFormatCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available format pairs were returned.

The number of format pairs supported **must** be greater than or equal to 1. `pSurfaceFormats` **must** not contain an entry whose value for `format` is `VK_FORMAT_UNDEFINED`.

If `pSurfaceFormats` includes an entry whose value for `colorSpace` is `VK_COLOR_SPACE_SRGB_NONLINEAR_KHR` and whose value for `format` is a UNORM (or SRGB) format and the corresponding SRGB (or UNORM) format is a color renderable format for `VK_IMAGE_TILING_OPTIMAL`, then `pSurfaceFormats` **must** also contain an entry with the same value for `colorSpace` and `format` equal to the corresponding SRGB (or UNORM) format.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDeviceSurfaceFormatsKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-surface-06524
surface **must** be a valid [VkSurfaceKHR](#) handle
- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-surface-06525
surface **must** be supported by **physicalDevice**, as reported by [vkGetPhysicalDeviceSurfaceSupportKHR](#) or an equivalent platform-specific mechanism

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-physicalDevice-parameter
physicalDevice **must** be a valid [VkPhysicalDevice](#) handle
- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-surface-parameter
If **surface** is not [VK_NULL_HANDLE](#), **surface** **must** be a valid [VkSurfaceKHR](#) handle
- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-pSurfaceFormatCount-parameter
pSurfaceFormatCount **must** be a valid pointer to a [uint32_t](#) value
- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-pSurfaceFormats-parameter
If the value referenced by **pSurfaceFormatCount** is not 0, and **pSurfaceFormats** is not [NULL](#), **pSurfaceFormats** **must** be a valid pointer to an array of **pSurfaceFormatCount** [VkSurfaceFormatKHR](#) structures
- VUID-vkGetPhysicalDeviceSurfaceFormatsKHR-commonparent
Both of **physicalDevice**, and **surface** that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same [VkInstance](#)

Return Codes

Success

- [VK_INCOMPLETE](#)
- [VK_SUCCESS](#)

Failure

- [VK_ERROR_OUT_OF_DEVICE_MEMORY](#)
- [VK_ERROR_OUT_OF_HOST_MEMORY](#)
- [VK_ERROR_SURFACE_LOST_KHR](#)
- [VK_ERROR_UNKNOWN](#)
- [VK_ERROR_VALIDATION_FAILED](#)

The [VkSurfaceFormatKHR](#) structure is defined as:

```
// Provided by VK_KHR_surface
typedef struct VkSurfaceFormatKHR {
    VkFormat          format;
    VkColorSpaceKHR   colorSpace;
} VkSurfaceFormatKHR;
```

- **format** is a [VkFormat](#) that is compatible with the specified surface.
- **colorSpace** is a presentation [VkColorSpaceKHR](#) that is compatible with the surface.

To query the supported swapchain format tuples for a surface, call:

```
// Provided by VK_KHR_get_surface_capabilities2
VkResult vkGetPhysicalDeviceSurfaceFormats2KHR(
    VkPhysicalDevice          physicalDevice,
    const VkPhysicalDeviceSurfaceInfo2KHR* pSurfaceInfo,
    uint32_t*                 pSurfaceFormatCount,
    VkSurfaceFormat2KHR*      pSurfaceFormats);
```

- **physicalDevice** is the physical device that will be associated with the swapchain to be created, as described for [vkCreateSwapchainKHR](#).
- **pSurfaceInfo** is a pointer to a [VkPhysicalDeviceSurfaceInfo2KHR](#) structure describing the surface and other fixed parameters that would be consumed by [vkCreateSwapchainKHR](#).
- **pSurfaceFormatCount** is a pointer to an integer related to the number of format tuples available or queried, as described below.
- **pSurfaceFormats** is either **NULL** or a pointer to an array of [VkSurfaceFormat2KHR](#) structures.

[vkGetPhysicalDeviceSurfaceFormats2KHR](#) behaves similarly to [vkGetPhysicalDeviceSurfaceFormatsKHR](#), with the ability to be extended via **pNext** chains.

If **pSurfaceFormats** is **NULL**, then the number of format tuples supported for the given **surface** is returned in **pSurfaceFormatCount**. Otherwise, **pSurfaceFormatCount** **must** point to a variable set by the application to the number of elements in the **pSurfaceFormats** array, and on return the variable is overwritten with the number of structures actually written to **pSurfaceFormats**. If the value of **pSurfaceFormatCount** is less than the number of format tuples supported, at most **pSurfaceFormatCount** structures will be written, and **VK_INCOMPLETE** will be returned instead of **VK_SUCCESS**, to indicate that not all the available values were returned.

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is **VK_TRUE**, [vkGetPhysicalDeviceSurfaceFormats2KHR](#) **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage

- VUID-vkGetPhysicalDeviceSurfaceFormats2KHR-pSurfaceInfo-06521
pSurfaceInfo->surface **must** be a valid [VkSurfaceKHR](#) handle
- VUID-vkGetPhysicalDeviceSurfaceFormats2KHR-pSurfaceInfo-06522

`pSurfaceInfo->surface` **must** be supported by `physicalDevice`, as reported by `vkGetPhysicalDeviceSurfaceSupportKHR` or an equivalent platform-specific mechanism

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfaceFormats2KHR-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceSurfaceFormats2KHR-pSurfaceInfo-parameter `pSurfaceInfo` **must** be a valid pointer to a valid `VkPhysicalDeviceSurfaceInfo2KHR` structure
- VUID-vkGetPhysicalDeviceSurfaceFormats2KHR-pSurfaceFormatCount-parameter `pSurfaceFormatCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceSurfaceFormats2KHR-pSurfaceFormats-parameter
If the value referenced by `pSurfaceFormatCount` is not 0, and `pSurfaceFormats` is not `NULL`, `pSurfaceFormats` **must** be a valid pointer to an array of `pSurfaceFormatCount` `VkSurfaceFormat2KHR` structures

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_SURFACE_LOST_KHR`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkSurfaceFormat2KHR` structure is defined as:

```
// Provided by VK_KHR_get_surface_capabilities2
typedef struct VkSurfaceFormat2KHR {
    VkStructureType    sType;
    void*              pNext;
    VkSurfaceFormatKHR surfaceFormat;
} VkSurfaceFormat2KHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- `surfaceFormat` is a `VkSurfaceFormatKHR` structure describing a format-color space pair that is compatible with the specified surface.

Valid Usage (Implicit)

- VUID-VkSurfaceFormat2KHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SURFACE_FORMAT_2_KHR`
- VUID-VkSurfaceFormat2KHR-pNext-pNext
`pNext` **must** be `NULL`

While the `format` of a presentable image refers to the encoding of each pixel, the `colorSpace` determines how the presentation engine interprets the pixel values. A color space in this document refers to a specific color space (defined by the chromaticities of its primaries and a white point in CIE Lab), and transfer functions indicating the mapping between the image data and the colorimetry with respect to the given color space.

Possible values of `VkSurfaceFormatKHR::colorSpace`, specifying the color spaces that a presentation engine can accept, are:

```

// Provided by VK_KHR_surface
typedef enum VkColorSpaceKHR {
    VK_COLOR_SPACE_SRGB_NONLINEAR_KHR = 0,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_DISPLAY_P3_NONLINEAR_EXT = 1000104001,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_EXTENDED_SRGB_LINEAR_EXT = 1000104002,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_DISPLAY_P3_LINEAR_EXT = 1000104003,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_DCI_P3_NONLINEAR_EXT = 1000104004,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_BT709_LINEAR_EXT = 1000104005,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_BT709_NONLINEAR_EXT = 1000104006,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_BT2020_LINEAR_EXT = 1000104007,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_HDR10_ST2084_EXT = 1000104008,
    // Provided by VK_EXT_swapchain_colorspace
    // VK_COLOR_SPACE_DOLBYVISION_EXT is legacy, but no reason was given in the API XML
    VK_COLOR_SPACE_DOLBYVISION_EXT = 1000104009,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_HDR10_HLG_EXT = 1000104010,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_ADOBERGB_LINEAR_EXT = 1000104011,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_ADOBERGB_NONLINEAR_EXT = 1000104012,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_PASS_THROUGH_EXT = 1000104013,
    // Provided by VK_EXT_swapchain_colorspace
    VK_COLOR_SPACE_EXTENDED_SRGB_NONLINEAR_EXT = 1000104014,
} VkColorSpaceKHR;

```

- **VK_COLOR_SPACE_SRGB_NONLINEAR_KHR** specifies support for the images in sRGB color space, encoded according to the sRGB specification.
- **VK_COLOR_SPACE_DISPLAY_P3_NONLINEAR_EXT** specifies support for the images in Display-P3 color space, encoded using a Display-P3 transfer function.
- **VK_COLOR_SPACE_EXTENDED_SRGB_LINEAR_EXT** specifies support for the images in extended sRGB color space, encoded using a linear transfer function.
- **VK_COLOR_SPACE_EXTENDED_SRGB_NONLINEAR_EXT** specifies support for the images in extended sRGB color space, encoded according to the scRGB specification.
- **VK_COLOR_SPACE_DISPLAY_P3_LINEAR_EXT** specifies support for the images in Display-P3 color space, encoded using a linear transfer function.
- **VK_COLOR_SPACE_DCI_P3_NONLINEAR_EXT** specifies support for the images in DCI-P3 color space, encoded according to the DCI-P3 specification. Note that values in such an image are interpreted as XYZ encoded color data by the presentation engine.

- `VK_COLOR_SPACE_BT709_LINEAR_EXT` specifies support for the images in BT709 color space, encoded using a linear transfer function.
- `VK_COLOR_SPACE_BT709_NONLINEAR_EXT` specifies support for the images in BT709 color space, encoded according to the BT709 specification.
- `VK_COLOR_SPACE_BT2020_LINEAR_EXT` specifies support for the images in BT2020 color space, encoded using a linear transfer function.
- `VK_COLOR_SPACE_HDR10_ST2084_EXT` specifies support for the images in HDR10 (BT2020) color space, encoded according to SMPTE ST2084 Perceptual Quantizer (PQ) specification.
- `VK_COLOR_SPACE_HDR10_HLG_EXT` specifies support for the images in HDR10 (BT2020) color space, encoded according to the Hybrid Log Gamma (HLG) specification.
- `VK_COLOR_SPACE_ADOBERGB_LINEAR_EXT` specifies support for images in Adobe RGB color space, encoded using a linear transfer function.
- `VK_COLOR_SPACE_ADOBERGB_NONLINEAR_EXT` specifies support for the images in Adobe RGB color space, encoded according to the Adobe RGB specification (approximately Gamma 2.2).
- `VK_COLOR_SPACE_PASS_THROUGH_EXT` specifies that color components are used “as is”. This is intended to allow applications to supply data for color spaces not described here.

NOTE

In older versions of the `VK_EXT_swapchain_colorspace` extension, `VK_COLOR_SPACE_DOLBYVISION_EXT` was exposed. The intent was to indicate the presentation engine shall decode an image using the SMPTE ST 2084 Perceptual Quantizer (PQ) EOTF, and then apply a proprietary OOTF to process the image. However, Dolby Vision profile 8.4 describes an encoding using the Hybrid Log Gamma (HLG) OETF, and there is no swapchain extension for signaling Dolby Vision metadata to be used by a proprietary OOTF. This enum is legacy but is maintained for backwards compatibility.

NOTE

For a traditional “Linear” or non-gamma transfer function color space use `VK_COLOR_SPACE_PASS_THROUGH_EXT`.

The presentation engine interprets the pixel values of the R, G, and B components as having been encoded using an appropriate transfer function. Applications **should** ensure that the appropriate transfer function has been applied. `Texel encode` requires that all implementations implicitly apply the sRGB EOTF¹ on R, G, and B components when shaders write to an sRGB pixel format image, which is useful for sRGB color spaces. For sRGB color spaces with other pixel formats, or other non-linear color spaces, applications **can** apply the transfer function explicitly in a shader. The A channel is always interpreted as linearly encoded.

This extension defines enums for `VkColorSpaceKHR` that correspond to the following color spaces:

Table 40. Color Spaces and Attributes

Name	Red Primary	Green Primary	Blue Primary	White-point	Transfer function
DCI-P3	1.000, 0.000	0.000, 1.000	0.000, 0.000	0.3333, 0.3333	DCI P3

Name	Red Primary	Green Primary	Blue Primary	White-point	Transfer function
Display-P3	0.680, 0.320	0.265, 0.690	0.150, 0.060	0.3127, 0.3290 (D65)	Display-P3
BT709	0.640, 0.330	0.300, 0.600	0.150, 0.060	0.3127, 0.3290 (D65)	BT709
sRGB	0.640, 0.330	0.300, 0.600	0.150, 0.060	0.3127, 0.3290 (D65)	sRGB
extended sRGB	0.640, 0.330	0.300, 0.600	0.150, 0.060	0.3127, 0.3290 (D65)	scRGB
HDR10_ST2084	0.708, 0.292	0.170, 0.797	0.131, 0.046	0.3127, 0.3290 (D65)	ST2084 PQ
HDR10_HLG	0.708, 0.292	0.170, 0.797	0.131, 0.046	0.3127, 0.3290 (D65)	HLG
Adobe RGB	0.640, 0.330	0.210, 0.710	0.150, 0.060	0.3127, 0.3290 (D65)	Adobe RGB

The transfer functions are described in the “Transfer Functions” chapter of the [Khronos Data Format Specification](#).

Except Display-P3 OETF, which is:

$$E = \begin{cases} 1.055 \times L^{\frac{1}{2.4}} - 0.055 & \text{for } 0.0030186 \leq L \leq 1 \\ 12.92 \times L & \text{for } 0 \leq L < 0.0030186 \end{cases}$$

where L is the linear value of a color component and E is the encoded value (as stored in the image in memory).

NOTE For most uses, the sRGB OETF is equivalent.

31.5.3. Surface Presentation Mode Support

To query the supported presentation modes for a surface, call:

```
// Provided by VK_KHR_surface
VkResult vkGetPhysicalDeviceSurfacePresentModesKHR(
    VkPhysicalDevice    physicalDevice,
    VkSurfaceKHR        surface,
    uint32_t*           pPresentModeCount,
    VkPresentModeKHR*   pPresentModes);
```

- **physicalDevice** is the physical device that will be associated with the swapchain to be created, as described for [vkCreateSwapchainKHR](#).

- **surface** is the surface that will be associated with the swapchain.
- **pPresentModeCount** is a pointer to an integer related to the number of presentation modes available or queried, as described below.
- **pPresentModes** is either **NULL** or a pointer to an array of **VkPresentModeKHR** values, indicating the supported presentation modes.

If **pPresentModes** is **NULL**, then the number of presentation modes supported for the given **surface** is returned in **pPresentModeCount**. Otherwise, **pPresentModeCount** **must** point to a variable set by the application to the number of elements in the **pPresentModes** array, and on return the variable is overwritten with the number of values actually written to **pPresentModes**. If the value of **pPresentModeCount** is less than the number of presentation modes supported, at most **pPresentModeCount** values will be written, and **VK_INCOMPLETE** will be returned instead of **VK_SUCCESS**, to indicate that not all the available modes were returned.

If **VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations** is **VK_TRUE**, **vkGetPhysicalDeviceSurfacePresentModesKHR** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage

- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-surface-06524
surface **must** be a valid **VkSurfaceKHR** handle
- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-surface-06525
surface **must** be supported by **physicalDevice**, as reported by **vkGetPhysicalDeviceSurfaceSupportKHR** or an equivalent platform-specific mechanism

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-physicalDevice-parameter
physicalDevice **must** be a valid **VkPhysicalDevice** handle
- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-surface-parameter
If **surface** is not **VK_NULL_HANDLE**, **surface** **must** be a valid **VkSurfaceKHR** handle
- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-pPresentModeCount-parameter
pPresentModeCount **must** be a valid pointer to a **uint32_t** value
- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-pPresentModes-parameter
If the value referenced by **pPresentModeCount** is not **0**, and **pPresentModes** is not **NULL**, **pPresentModes** **must** be a valid pointer to an array of **pPresentModeCount** **VkPresentModeKHR** values
- VUID-vkGetPhysicalDeviceSurfacePresentModesKHR-commonparent
Both of **physicalDevice**, and **surface** that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same **VkInstance**

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_SURFACE_LOST_KHR`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

Possible values of elements of the `vkGetPhysicalDeviceSurfacePresentModesKHR::pPresentModes` array, indicating the supported presentation modes for a surface, are:

```
// Provided by VK_KHR_surface
typedef enum VkPresentModeKHR {
    VK_PRESENT_MODE_IMMEDIATE_KHR = 0,
    VK_PRESENT_MODE_MAILBOX_KHR = 1,
    VK_PRESENT_MODE_FIFO_KHR = 2,
    VK_PRESENT_MODE_FIFO_RELAXED_KHR = 3,
    // Provided by VK_KHR_shared_presentable_image
    VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR = 1000111000,
    // Provided by VK_KHR_shared_presentable_image
    VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR = 1000111001,
} VkPresentModeKHR;
```

- `VK_PRESENT_MODE_IMMEDIATE_KHR` specifies that the presentation engine does not wait for a vertical blanking period to update the current image, meaning this mode **may** result in visible tearing. No internal queuing of presentation requests is needed, as the requests are applied immediately.
- `VK_PRESENT_MODE_MAILBOX_KHR` specifies that the presentation engine waits for the next vertical blanking period to update the current image. Tearing **cannot** be observed. An internal single-entry queue is used to hold pending presentation requests. If the queue is full when a new presentation request is received, the new request replaces the existing entry, and any images associated with the prior entry become available for reuse by the application. One request is removed from the queue and processed during each vertical blanking period in which the queue is non-empty.
- `VK_PRESENT_MODE_FIFO_KHR` specifies that the presentation engine waits for the next vertical blanking period to update the current image. Tearing **cannot** be observed. An internal queue is used to hold pending presentation requests. New requests are appended to the end of the queue, and one request is removed from the beginning of the queue and processed during each vertical blanking period in which the queue is non-empty. This is the only value of `presentMode`

that is **required** to be supported.

- **VK_PRESENT_MODE_FIFO_RELAXED_KHR** specifies that the presentation engine generally waits for the next vertical blanking period to update the current image. If a vertical blanking period has already passed since the last update of the current image then the presentation engine does not wait for another vertical blanking period for the update, meaning this mode **may** result in visible tearing in this case. This mode is useful for reducing visual stutter with an application that will mostly present a new image before the next vertical blanking period, but may occasionally be late, and present a new image just after the next vertical blanking period. An internal queue is used to hold pending presentation requests. New requests are appended to the end of the queue, and one request is removed from the beginning of the queue and processed during or after each vertical blanking period in which the queue is non-empty.
- **VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR** specifies that the presentation engine and application have concurrent access to a single image, which is referred to as a *shared presentable image*. The presentation engine is only required to update the current image after a new presentation request is received. Therefore the application **must** make a presentation request whenever an update is required. However, the presentation engine **may** update the current image at any point, meaning this mode **may** result in visible tearing.
- **VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR** specifies that the presentation engine and application have concurrent access to a single image, which is referred to as a *shared presentable image*. The presentation engine periodically updates the current image on its regular refresh cycle. The application is only required to make one initial presentation request, after which the presentation engine **must** update the current image without any need for further presentation requests. The application **can** indicate the image contents have been updated by making a presentation request, but this does not guarantee the timing of when it will be updated. This mode **may** result in visible tearing if rendering to the image is not timed correctly.

The supported **VkImageUsageFlagBits** of the presentable images of a swapchain created for a surface **may** differ depending on the presentation mode, and can be determined as per the table below:

Table 41. Presentable Image Usage Queries

Presentation mode	Image usage flags
VK_PRESENT_MODE_IMMEDIATE_KHR	VkSurfaceCapabilitiesKHR::supportedUsageFlags
VK_PRESENT_MODE_MAILBOX_KHR	VkSurfaceCapabilitiesKHR::supportedUsageFlags
VK_PRESENT_MODE_FIFO_KHR	VkSurfaceCapabilitiesKHR::supportedUsageFlags
VK_PRESENT_MODE_FIFO_RELAXED_KHR	VkSurfaceCapabilitiesKHR::supportedUsageFlags
VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR	VkSharedPresentSurfaceCapabilitiesKHR::sharedPresentSupportedUsageFlags
VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR	VkSharedPresentSurfaceCapabilitiesKHR::sharedPresentSupportedUsageFlags

NOTE

For reference, the mode indicated by **VK_PRESENT_MODE_FIFO_KHR** is equivalent to the behavior of {wgl|glX|egl}SwapBuffers with a swap interval of 1, while the mode

indicated by `VK_PRESENT_MODE_FIFO_RELAXED_KHR` is equivalent to the behavior of `{wgl|glX}SwapBuffers` with a swap interval of -1 (from the `{WGL|GLX}_EXT_swap_control_tear` extensions).

31.6. Device Group Queries

A logical device that represents multiple physical devices **may** support presenting from images on more than one physical device, or combining images from multiple physical devices.

To query these capabilities, call:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
VkResult vkGetDeviceGroupPresentCapabilitiesKHR(
    VkDevice device,
    VkDeviceGroupPresentCapabilitiesKHR* pDeviceGroupPresentCapabilities);
```

- `device` is the logical device.
- `pDeviceGroupPresentCapabilities` is a pointer to a `VkDeviceGroupPresentCapabilitiesKHR` structure in which the device's capabilities are returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetDeviceGroupPresentCapabilitiesKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetDeviceGroupPresentCapabilitiesKHR-device-parameter `device` **must** be a valid `VkDevice` handle
- VUID-vkGetDeviceGroupPresentCapabilitiesKHR-pDeviceGroupPresentCapabilities-parameter `pDeviceGroupPresentCapabilities` **must** be a valid pointer to a `VkDeviceGroupPresentCapabilitiesKHR` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkDeviceGroupPresentCapabilitiesKHR` structure is defined as:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
typedef struct VkDeviceGroupPresentCapabilitiesKHR {
    VkStructureType    sType;
    void*              pNext;
    uint32_t            presentMask[VK_MAX_DEVICE_GROUP_SIZE];
    VkDeviceGroupPresentModeFlagsKHR    modes;
} VkDeviceGroupPresentCapabilitiesKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `presentMask` is an array of `VK_MAX_DEVICE_GROUP_SIZE` `uint32_t` masks, where the mask at element `i` is non-zero if physical device `i` has a presentation engine, and where bit `j` is set in element `i` if physical device `i` **can** present swapchain images from physical device `j`. If element `i` is non-zero, then bit `i` **must** be set.
- `modes` is a bitmask of `VkDeviceGroupPresentModeFlagBitsKHR` indicating which device group presentation modes are supported.

`modes` always has `VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR` set.

The present mode flags are also used when presenting an image, in `VkDeviceGroupPresentInfoKHR::mode`.

If a device group only includes a single physical device, then `modes` **must** equal `VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR`.

Valid Usage (Implicit)

- VUID-VkDeviceGroupPresentCapabilitiesKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_CAPABILITIES_KHR`
- VUID-VkDeviceGroupPresentCapabilitiesKHR-pNext-pNext
`pNext` **must** be `NULL`

Bits which **may** be set in `VkDeviceGroupPresentCapabilitiesKHR::modes`, indicating which device group presentation modes are supported, are:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
typedef enum VkDeviceGroupPresentModeFlagBitsKHR {
    VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR = 0x00000001,
    VK_DEVICE_GROUP_PRESENT_MODE_REMOTE_BIT_KHR = 0x00000002,
    VK_DEVICE_GROUP_PRESENT_MODE_SUM_BIT_KHR = 0x00000004,
    VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_MULTI_DEVICE_BIT_KHR = 0x00000008,
} VkDeviceGroupPresentModeFlagBitsKHR;
```

- `VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR` specifies that any physical device with a presentation engine **can** present its own swapchain images.
- `VK_DEVICE_GROUP_PRESENT_MODE_REMOTE_BIT_KHR` specifies that any physical device with a presentation engine **can** present swapchain images from any physical device in its `presentMask`.
- `VK_DEVICE_GROUP_PRESENT_MODE_SUM_BIT_KHR` specifies that any physical device with a presentation engine **can** present the sum of swapchain images from any physical devices in its `presentMask`.
- `VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_MULTI_DEVICE_BIT_KHR` specifies that multiple physical devices with a presentation engine **can** each present their own swapchain images.

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
typedef VkFlags VkDeviceGroupPresentModeFlagsKHR;
```

`VkDeviceGroupPresentModeFlagsKHR` is a bitmask type for setting a mask of zero or more `VkDeviceGroupPresentModeFlagBitsKHR`.

Some surfaces **may** not be capable of using all the device group present modes.

To query the supported device group present modes for a particular surface, call:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
VkResult vkGetDeviceGroupSurfacePresentModesKHR(
    VkDevice          device,
    VkSurfaceKHR      surface,
    VkDeviceGroupPresentModeFlagsKHR* pModes);
```

- `device` is the logical device.
- `surface` is the surface.
- `pModes` is a pointer to a `VkDeviceGroupPresentModeFlagsKHR` in which the supported device group present modes for the surface are returned.

The modes returned by this command are not invariant, and **may** change in response to the surface being moved, resized, or occluded. These modes **must** be a subset of the modes returned by `vkGetDeviceGroupPresentCapabilitiesKHR`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetDeviceGroupSurfacePresentModesKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetDeviceGroupSurfacePresentModesKHR-surface-06212
`surface` **must** be supported by all physical devices associated with `device`, as reported by `vkGetPhysicalDeviceSurfaceSupportKHR` or an equivalent platform-specific mechanism

Valid Usage (Implicit)

- VUID-vkGetDeviceGroupSurfacePresentModesKHR-device-parameter **device** **must** be a valid [VkDevice](#) handle
- VUID-vkGetDeviceGroupSurfacePresentModesKHR-surface-parameter **surface** **must** be a valid [VkSurfaceKHR](#) handle
- VUID-vkGetDeviceGroupSurfacePresentModesKHR-pModes-parameter **pModes** **must** be a valid pointer to a [VkDeviceGroupPresentModeFlagsKHR](#) value
- VUID-vkGetDeviceGroupSurfacePresentModesKHR-commonparent Both of **device**, and **surface** **must** have been created, allocated, or retrieved from the same [VkInstance](#)

Host Synchronization

- Host access to **surface** **must** be externally synchronized

Return Codes

Success

- [VK_SUCCESS](#)

Failure

- [VK_ERROR_OUT_OF_DEVICE_MEMORY](#)
- [VK_ERROR_OUT_OF_HOST_MEMORY](#)
- [VK_ERROR_SURFACE_LOST_KHR](#)
- [VK_ERROR_UNKNOWN](#)
- [VK_ERROR_VALIDATION_FAILED](#)

When using [VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_MULTI_DEVICE_BIT_KHR](#), the application **may** need to know which regions of the surface are used when presenting locally on each physical device. Presentation of swapchain images to this surface need only have valid contents in the regions returned by this command.

To query a set of rectangles used in presentation on the physical device, call:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
VkResult vkGetPhysicalDevicePresentRectanglesKHR(
    VkPhysicalDevice      physicalDevice,
    VkSurfaceKHR          surface,
    uint32_t*             pRectCount,
    VkRect2D*             pRects);
```

- `physicalDevice` is the physical device.
- `surface` is the surface.
- `pRectCount` is a pointer to an integer related to the number of rectangles available or queried, as described below.
- `pRects` is either `NULL` or a pointer to an array of `VkRect2D` structures.

If `pRects` is `NULL`, then the number of rectangles used when presenting the given `surface` is returned in `pRectCount`. Otherwise, `pRectCount` **must** point to a variable set by the application to the number of elements in the `pRects` array, and on return the variable is overwritten with the number of structures actually written to `pRects`. If the value of `pRectCount` is less than the number of rectangles, at most `pRectCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available rectangles were returned.

The values returned by this command are not invariant, and **may** change in response to the surface being moved, resized, or occluded.

The rectangles returned by this command **must** not overlap.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDevicePresentRectanglesKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetPhysicalDevicePresentRectanglesKHR-surface-06211
`surface` **must** be supported by `physicalDevice`, as reported by `vkGetPhysicalDeviceSurfaceSupportKHR` or an equivalent platform-specific mechanism

Valid Usage (Implicit)

- VUID-vkGetPhysicalDevicePresentRectanglesKHR-physicalDevice-parameter
`physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDevicePresentRectanglesKHR-surface-parameter
`surface` **must** be a valid `VkSurfaceKHR` handle
- VUID-vkGetPhysicalDevicePresentRectanglesKHR-pRectCount-parameter
`pRectCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDevicePresentRectanglesKHR-pRects-parameter
If the value referenced by `pRectCount` is not `0`, and `pRects` is not `NULL`, `pRects` **must** be a valid pointer to an array of `pRectCount` `VkRect2D` structures
- VUID-vkGetPhysicalDevicePresentRectanglesKHR-commonparent
Both of `physicalDevice`, and `surface` **must** have been created, allocated, or retrieved from the same `VkInstance`

Host Synchronization

- Host access to **surface** **must** be externally synchronized

Return Codes

Success

- **VK_INCOMPLETE**
- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

31.7. WSI Swapchain

A swapchain object (a.k.a. swapchain) provides the ability to present rendering results to a surface. Swapchain objects are represented by **VkSwapchainKHR** handles:

```
// Provided by VK_KHR_swapchain
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSwapchainKHR)
```

A swapchain is an abstraction for an array of presentable images that are associated with a surface. The presentable images are represented by **VkImage** objects created by the platform. One image (which **can** be an array image for multiview/stereoscopic-3D surfaces) is displayed at a time, but multiple images **can** be queued for presentation. An application renders to the image, and then queues the image for presentation to the surface.

A native window **cannot** be associated with more than one non-retired swapchain at a time. Further, swapchains **cannot** be created for native windows that have a non-Vulkan graphics API surface associated with them.

NOTE

The presentation engine is an abstraction for the platform's compositor or display engine.

The presentation engine **may** be synchronous or asynchronous with respect to the application and/or logical device.

Some implementations **may** use the device's graphics queue or dedicated presentation hardware to perform presentation.

The presentable images of a swapchain are owned by the presentation engine. An application **can** acquire use of a presentable image from the presentation engine. Use of a presentable image **must** occur only after the image is returned by `vkAcquireNextImageKHR`, and before it is released by `vkQueuePresentKHR`. This includes transitioning the image layout and rendering commands.

An application **can** acquire use of a presentable image with `vkAcquireNextImageKHR`. After acquiring a presentable image and before modifying it, the application **must** use a synchronization primitive to ensure that the presentation engine has finished reading from the image. The application **can** then transition the image's layout, queue rendering commands to it, etc. Finally, the application presents the image with `vkQueuePresentKHR`, which releases the acquisition of the image.

The presentation engine controls the order in which presentable images are acquired for use by the application.

NOTE

This allows the platform to handle situations which require out-of-order return of images after presentation. At the same time, it allows the application to generate command buffers referencing all of the images in the swapchain at initialization time, rather than in its main loop.

Swapchains **cannot** be destroyed [SCID-4]. If `VkPhysicalDeviceVulkanSC10Properties::deviceDestroyFreesMemory` is `VK_TRUE`, the memory for swapchain images is returned to the system when the device is destroyed. Otherwise, it **may** not be returned to the system until the process is terminated.

How this all works is described below.

If a swapchain is created with `presentMode` set to either `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` or `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`, a single presentable image **can** be acquired, referred to as a shared presentable image. A shared presentable image **may** be concurrently accessed by the application and the presentation engine, without transitioning the image's layout after it is initially presented.

- With `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR`, the presentation engine is only required to update to the latest contents of a shared presentable image after a present. The application **must** call `vkQueuePresentKHR` to guarantee an update. However, the presentation engine **may** update from it at any time.
- With `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`, the presentation engine will automatically present the latest contents of a shared presentable image during every refresh cycle. The application is only required to make one initial call to `vkQueuePresentKHR`, after which the presentation engine will update from it without any need for further present calls. The application **can** indicate the image contents have been updated by calling `vkQueuePresentKHR`, but this does not guarantee the timing of when updates will occur.

The presentation engine **may** access a shared presentable image at any time after it is first presented. To avoid tearing, an application **should** coordinate access with the presentation engine. This requires presentation engine timing information through platform-specific mechanisms and ensuring that color attachment writes are made available during the portion of the presentation engine's refresh cycle they are intended for.

NOTE

The [VK_KHR_shared_presentable_image](#) extension does not provide functionality for determining the timing of the presentation engine's refresh cycles.

In order to query a swapchain's status when rendering to a shared presentable image, call:

```
// Provided by VK_KHR_shared_presentable_image
VkResult vkGetSwapchainStatusKHR(
    VkDevice          device,
    VkSwapchainKHR    swapchain);
```

- **device** is the device associated with **swapchain**.
- **swapchain** is the swapchain to query.

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is **VK_TRUE**, **vkGetSwapchainStatusKHR** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage (Implicit)

- VUID-vkGetSwapchainStatusKHR-device-parameter **device** **must** be a valid [VkDevice](#) handle
- VUID-vkGetSwapchainStatusKHR-swapchain-parameter **swapchain** **must** be a valid [VkSwapchainKHR](#) handle
- VUID-vkGetSwapchainStatusKHR-swapchain-parent **swapchain** **must** have been created, allocated, or retrieved from **device**

Host Synchronization

- Host access to **swapchain** **must** be externally synchronized

Return Codes

Success

- **VK_SUBOPTIMAL_KHR**
- **VK_SUCCESS**

Failure

- **VK_ERROR_DEVICE_LOST**
- **VK_ERROR_OUT_OF_DATE_KHR**
- **VK_ERROR_OUT_OF_DEVICE_MEMORY**
- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_SURFACE_LOST_KHR**

- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The possible return values for `vkGetSwapchainStatusKHR` **should** be interpreted as follows:

- `VK_SUCCESS` specifies the presentation engine is presenting the contents of the shared presentable image, as per the swapchain's `VkPresentModeKHR`.
- `VK_SUBOPTIMAL_KHR` the swapchain no longer matches the surface properties exactly, but the presentation engine is presenting the contents of the shared presentable image, as per the swapchain's `VkPresentModeKHR`.
- `VK_ERROR_OUT_OF_DATE_KHR` the surface has changed in such a way that it is no longer compatible with the swapchain.
- `VK_ERROR_SURFACE_LOST_KHR` the surface is no longer available.

NOTE

The swapchain state **may** be cached by implementations, so applications **should** regularly call `vkGetSwapchainStatusKHR` when using a swapchain with `VkPresentModeKHR` equal to `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`.

To create a swapchain, call:

```
// Provided by VK_KHR_swapchain
VkResult vkCreateSwapchainKHR(
    VkDevice                                device,
    const VkSwapchainCreateInfoKHR*         pCreateInfo,
    const VkAllocationCallbacks*            pAllocator,
    VkSwapchainKHR*                         pSwapchain);
```

- `device` is the device to create the swapchain for.
- `pCreateInfo` is a pointer to a `VkSwapchainCreateInfoKHR` structure specifying the parameters of the created swapchain.
- `pAllocator` is the allocator used for host memory allocated for the swapchain object when there is no more specific allocator available (see [Memory Allocation](#)).
- `pSwapchain` is a pointer to a `VkSwapchainKHR` handle in which the created swapchain object will be returned.

As mentioned above, if `vkCreateSwapchainKHR` succeeds, it will return a handle to a swapchain containing an array of at least `pCreateInfo->minImageCount` presentable images.

While acquired by the application, presentable images **can** be used in any way that equivalent non-presentable images **can** be used. A presentable image is equivalent to a non-presentable image created with the following `VkImageCreateInfo` parameters:

VkImageCreateInfo Field	Value
flags	VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT is set if VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR is set VK_IMAGE_CREATE_PROTECTED_BIT is set if VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR is set VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT and VK_IMAGE_CREATE_EXTENDED_USAGE_BIT_KHR are both set if VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR is set all other bits are unset
imageType	VK_IMAGE_TYPE_2D
format	pCreateInfo->imageFormat
extent	{pCreateInfo->imageExtent.width, pCreateInfo->imageExtent.height, 1}
mipLevels	1
arrayLayers	pCreateInfo->imageArrayLayers
samples	VK_SAMPLE_COUNT_1_BIT
tiling	VK_IMAGE_TILING_OPTIMAL
usage	pCreateInfo->imageUsage
sharingMode	pCreateInfo->imageSharingMode
queueFamilyIndexCount	pCreateInfo->queueFamilyIndexCount
pQueueFamilyIndices	pCreateInfo->pQueueFamilyIndices
initialLayout	VK_IMAGE_LAYOUT_UNDEFINED

The `pCreateInfo->surface` **must** not be destroyed until after the swapchain is destroyed.

If the native window referred to by `pCreateInfo->surface` is already associated with a Vulkan swapchain, `VK_ERROR_NATIVE_WINDOW_IN_USE_KHR` **must** be returned.

If the native window referred to by `pCreateInfo->surface` is already associated with a non-Vulkan graphics API surface, `VK_ERROR_NATIVE_WINDOW_IN_USE_KHR` **must** be returned.

The native window referred to by `pCreateInfo->surface` **must** not become associated with a non-Vulkan graphics API surface before all associated Vulkan swapchains have been destroyed.

`vkCreateSwapchainKHR` will return `VK_ERROR_DEVICE_LOST` if the logical device was lost. However, `VkSurfaceKHR` is not a child of any `VkDevice` and is not affected by the lost device. After successfully recreating a `VkDevice`, the same `VkSurfaceKHR` **can** be used to create a new `VkSwapchainKHR`, provided the previous one was destroyed.

When the [VkSurfaceKHR](#) in [VkSwapchainCreateInfoKHR](#) is a display surface, then the [VkDisplayModeKHR](#) in display surface's [VkDisplaySurfaceCreateInfoKHR](#) is associated with a particular [VkDisplayKHR](#). Swapchain creation **may** fail if that [VkDisplayKHR](#) is not acquired by the application. In this scenario [VK_ERROR_INITIALIZATION_FAILED](#) is returned.

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is [VK_TRUE](#), [vkCreateSwapchainKHR](#) **must** not return [VK_ERROR_OUT_OF_HOST_MEMORY](#).

Valid Usage

- VUID-vkCreateSwapchainKHR-device-05068

The number of swapchains currently allocated from [device](#) plus 1 **must** be less than or equal to the total number of swapchains requested via [VkDeviceObjectReservationCreateInfo::swapchainRequestCount](#) specified when [device](#) was created

Valid Usage (Implicit)

- VUID-vkCreateSwapchainKHR-device-parameter
[device](#) **must** be a valid [VkDevice](#) handle
- VUID-vkCreateSwapchainKHR-pCreateInfo-parameter
[pCreateInfo](#) **must** be a valid pointer to a valid [VkSwapchainCreateInfoKHR](#) structure
- VUID-vkCreateSwapchainKHR-pAllocator-null
[pAllocator](#) **must** be [NULL](#)
- VUID-vkCreateSwapchainKHR-pSwapchain-parameter
[pSwapchain](#) **must** be a valid pointer to a [VkSwapchainKHR](#) handle
- VUID-vkCreateSwapchainKHR-device-queuecount
The device **must** have been created with at least [1](#) queue

Return Codes

Success

- [VK_SUCCESS](#)

Failure

- [VK_ERROR_DEVICE_LOST](#)
- [VK_ERROR_INITIALIZATION_FAILED](#)
- [VK_ERROR_NATIVE_WINDOW_IN_USE_KHR](#)
- [VK_ERROR_OUT_OF_DEVICE_MEMORY](#)
- [VK_ERROR_OUT_OF_HOST_MEMORY](#)
- [VK_ERROR_SURFACE_LOST_KHR](#)
- [VK_ERROR_UNKNOWN](#)

- `VK_ERROR_VALIDATION_FAILED`

The `VkSwapchainCreateInfoKHR` structure is defined as:

```
// Provided by VK_KHR_swapchain
typedef struct VkSwapchainCreateInfoKHR {
    VkStructureType           sType;
    const void*               pNext;
    VkSwapchainCreateFlagsKHR flags;
    VkSurfaceKHR              surface;
    uint32_t                  minImageCount;
    VkFormat                  imageFormat;
    VkColorSpaceKHR           imageColorSpace;
    VkExtent2D                imageExtent;
    uint32_t                  imageArrayLayers;
    VkImageUsageFlags         imageUsage;
    VkSharingMode              imageSharingMode;
    uint32_t                  queueFamilyIndexCount;
    const uint32_t*           pQueueFamilyIndices;
    VkSurfaceTransformFlagBitsKHR preTransform;
    VkCompositeAlphaFlagBitsKHR compositeAlpha;
    VkPresentModeKHR          presentMode;
    VkBool32                  clipped;
    VkSwapchainKHR            oldSwapchain;
} VkSwapchainCreateInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is a bitmask of `VkSwapchainCreateFlagBitsKHR` indicating parameters of the swapchain creation.
- `surface` is the surface onto which the swapchain will present images. If the creation succeeds, the swapchain becomes associated with `surface`.
- `minImageCount` is the minimum number of presentable images that the application needs. The implementation will either create the swapchain with at least that many images, or it will fail to create the swapchain.
- `imageFormat` is a `VkFormat` value specifying the format the swapchain image(s) will be created with.
- `imageColorSpace` is a `VkColorSpaceKHR` value specifying the way the swapchain interprets image data.
- `imageExtent` is the size (in pixels) of the swapchain image(s). The behavior is platform-dependent if the image extent does not match the surface's `currentExtent` as returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR`.

NOTE

On some platforms, it is normal that `maxImageExtent` may become `(0, 0)`, for example when the window is minimized. In such a case, it is not possible to

create a swapchain due to the Valid Usage requirements .

- **imageArrayLayers** is the number of views in a multiview/stereo surface. For non-stereoscopic-3D applications, this value is 1.
- **imageUsage** is a bitmask of **VkImageUsageFlagBits** describing the intended usage of the (acquired) swapchain images.
- **imageSharingMode** is the sharing mode used for the image(s) of the swapchain.
- **queueFamilyIndexCount** is the number of queue families having access to the image(s) of the swapchain when **imageSharingMode** is **VK_SHARING_MODE_CONCURRENT**.
- **pQueueFamilyIndices** is a pointer to an array of queue family indices having access to the images(s) of the swapchain when **imageSharingMode** is **VK_SHARING_MODE_CONCURRENT**.
- **preTransform** is a **VkSurfaceTransformFlagBitsKHR** value describing the transform, relative to the presentation engine's natural orientation, applied to the image content prior to presentation. If it does not match the **currentTransform** value returned by **vkGetPhysicalDeviceSurfaceCapabilitiesKHR**, the presentation engine will transform the image content as part of the presentation operation.
- **compositeAlpha** is a **VkCompositeAlphaFlagBitsKHR** value indicating the alpha compositing mode to use when this surface is composited together with other surfaces on certain window systems.
- **presentMode** is the presentation mode the swapchain will use. A swapchain's present mode determines how incoming present requests will be processed and queued internally.
- **clipped** specifies whether the Vulkan implementation is allowed to discard rendering operations that affect regions of the surface that are not visible.
 - If **clipped** is **VK_TRUE**, the presentable images associated with the swapchain **may** not own all of their pixels. Pixels in the presentable images that correspond to regions of the target surface obscured by another window on the desktop, or subject to some other clipping mechanism will have undefined content when read back. Fragment shaders **may** not execute for these pixels, and thus any side effects they would have had will not occur. Setting **VK_TRUE** does not guarantee any clipping will occur, but allows more efficient presentation methods to be used on some platforms.
 - If **clipped** is **VK_FALSE**, presentable images associated with the swapchain will own all of the pixels they contain.

NOTE

Applications **should** set this value to **VK_TRUE** if they do not expect to read back the content of presentable images before presenting them or after reacquiring them, and if their fragment shaders do not have any side effects that require them to run for all pixels in the presentable image.

- **oldSwapchain** **must** be **VK_NULL_HANDLE** in Vulkan SC [SCID-4].

Valid Usage

- VUID-VkSwapchainCreateInfoKHR-surface-01270
surface **must** be a surface that is supported by the device as determined using

vkGetPhysicalDeviceSurfaceSupportKHR

- VUID-VkSwapchainCreateInfoKHR-minImageCount-01272
`minImageCount` **must** be less than or equal to the value returned in the `maxImageCount` member of the `VkSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` for the surface if the returned `maxImageCount` is not zero
- VUID-VkSwapchainCreateInfoKHR-presentMode-02839
If `presentMode` is not `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` nor `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`, then `minImageCount` **must** be greater than or equal to the value returned in the `minImageCount` member of the `VkSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` for the surface
- VUID-VkSwapchainCreateInfoKHR-minImageCount-01383
`minImageCount` **must** be 1 if `presentMode` is either `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` or `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`
- VUID-VkSwapchainCreateInfoKHR-imageFormat-01273
`imageFormat` and `imageColorSpace` **must** match the `format` and `colorSpace` members, respectively, of one of the `VkSurfaceFormatKHR` structures returned by `vkGetPhysicalDeviceSurfaceFormatsKHR` for the surface
- VUID-VkSwapchainCreateInfoKHR-pNext-07781
`imageExtent` **must** be between `minImageExtent` and `maxImageExtent`, inclusive, where `minImageExtent` and `maxImageExtent` are members of the `VkSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` for the surface
- VUID-VkSwapchainCreateInfoKHR-imageExtent-01689
`imageExtent` members `width` and `height` **must** both be non-zero
- VUID-VkSwapchainCreateInfoKHR-imageArrayLayers-01275
`imageArrayLayers` **must** be greater than 0 and less than or equal to the `maxImageArrayLayers` member of the `VkSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` for the surface
- VUID-VkSwapchainCreateInfoKHR-presentMode-01427
If `presentMode` is `VK_PRESENT_MODE_IMMEDIATE_KHR`, `VK_PRESENT_MODE_MAILBOX_KHR`, `VK_PRESENT_MODE_FIFO_KHR` or `VK_PRESENT_MODE_FIFO_RELAXED_KHR`, `imageUsage` **must** be a subset of the supported usage flags present in the `supportedUsageFlags` member of the `VkSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` for surface
- VUID-VkSwapchainCreateInfoKHR-imageUsage-01384
If `presentMode` is `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` or `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`, `imageUsage` **must** be a subset of the supported usage flags present in the `sharedPresentSupportedUsageFlags` member of the `VkSharedPresentSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilities2KHR` for surface
- VUID-VkSwapchainCreateInfoKHR-imageSharingMode-01277
If `imageSharingMode` is `VK_SHARING_MODE_CONCURRENT`, `pQueueFamilyIndices` **must** be a valid

pointer to an array of `queueFamilyIndexCount` `uint32_t` values

- VUID-VkSwapchainCreateInfoKHR-imageSharingMode-01278

If `imageSharingMode` is `VK_SHARING_MODE_CONCURRENT`, `queueFamilyIndexCount` **must** be greater than 1

- VUID-VkSwapchainCreateInfoKHR-imageSharingMode-01428

If `imageSharingMode` is `VK_SHARING_MODE_CONCURRENT`, each element of `pQueueFamilyIndices` **must** be unique and **must** be less than `pQueueFamilyPropertyCount` returned by either `vkGetPhysicalDeviceQueueFamilyProperties` or `vkGetPhysicalDeviceQueueFamilyProperties2` for the `physicalDevice` that was used to create `device`

- VUID-VkSwapchainCreateInfoKHR-preTransform-01279

`preTransform` **must** be one of the bits present in the `supportedTransforms` member of the `VkSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` for the surface

- VUID-VkSwapchainCreateInfoKHR-compositeAlpha-01280

`compositeAlpha` **must** be one of the bits present in the `supportedCompositeAlpha` member of the `VkSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` for the surface

- VUID-VkSwapchainCreateInfoKHR-presentMode-01281

`presentMode` **must** be one of the `VkPresentModeKHR` values returned by `vkGetPhysicalDeviceSurfacePresentModesKHR` for the surface

- VUID-VkSwapchainCreateInfoKHR-physicalDeviceCount-01429

`flags` **must** not contain `VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR`

- VUID-VkSwapchainCreateInfoKHR-oldSwapchain-05073

`oldSwapchain` **must** be `VK_NULL_HANDLE`

- VUID-VkSwapchainCreateInfoKHR-imageFormat-01778

The `implied image creation parameters` of the swapchain **must** be supported as reported by `vkGetPhysicalDeviceImageFormatProperties`

- VUID-VkSwapchainCreateInfoKHR-flags-03168

If `flags` contains `VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR` then the `pNext` chain **must** include a `VkImageFormatListCreateInfo` structure with a `viewFormatCount` greater than zero and `pViewFormats` **must** have an element equal to `imageFormat`

- VUID-VkSwapchainCreateInfoKHR-pNext-04099

If a `VkImageFormatListCreateInfo` structure was included in the `pNext` chain and `VkImageFormatListCreateInfo::viewFormatCount` is not zero then all of the formats in `VkImageFormatListCreateInfo::pViewFormats` **must** be compatible with the `format` as described in the `compatibility table`

- VUID-VkSwapchainCreateInfoKHR-flags-04100

If `flags` does not contain `VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR` and the `pNext` chain include a `VkImageFormatListCreateInfo` structure then `VkImageFormatListCreateInfo::viewFormatCount` **must** be 0 or 1

Valid Usage (Implicit)

- VUID-VkSwapchainCreateInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR`
- VUID-VkSwapchainCreateInfoKHR-pNext-pNext
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkDeviceGroupSwapchainCreateInfoKHR`, `VkImageFormatListCreateInfo`, or `VkSwapchainCounterCreateInfoEXT`
- VUID-VkSwapchainCreateInfoKHR-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique
- VUID-VkSwapchainCreateInfoKHR-flags-parameter
`flags` **must** be a valid combination of `VkSwapchainCreateFlagBitsKHR` values
- VUID-VkSwapchainCreateInfoKHR-surface-parameter
`surface` **must** be a valid `VkSurfaceKHR` handle
- VUID-VkSwapchainCreateInfoKHR-imageFormat-parameter
`imageFormat` **must** be a valid `VkFormat` value
- VUID-VkSwapchainCreateInfoKHR-imageColorSpace-parameter
`imageColorSpace` **must** be a valid `VkColorSpaceKHR` value
- VUID-VkSwapchainCreateInfoKHR-imageUsage-parameter
`imageUsage` **must** be a valid combination of `VkImageUsageFlagBits` values
- VUID-VkSwapchainCreateInfoKHR-imageUsage-requiredbitmask
`imageUsage` **must** not be `0`
- VUID-VkSwapchainCreateInfoKHR-imageSharingMode-parameter
`imageSharingMode` **must** be a valid `VkSharingMode` value
- VUID-VkSwapchainCreateInfoKHR-preTransform-parameter
`preTransform` **must** be a valid `VkSurfaceTransformFlagBitsKHR` value
- VUID-VkSwapchainCreateInfoKHR-compositeAlpha-parameter
`compositeAlpha` **must** be a valid `VkCompositeAlphaFlagBitsKHR` value
- VUID-VkSwapchainCreateInfoKHR-presentMode-parameter
`presentMode` **must** be a valid `VkPresentModeKHR` value
- VUID-VkSwapchainCreateInfoKHR-commonparent
Both of `oldSwapchain`, and `surface` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same `VkInstance`

Host Synchronization

- Host access to `surface` **must** be externally synchronized

Bits which **can** be set in `VkSwapchainCreateInfoKHR::flags`, specifying parameters of swapchain creation, are:


```
// Provided by VK_KHR_swapchain
typedef enum VkSwapchainCreateFlagBitsKHR {
    // Provided by VK_VERSION_1_1 with VK_KHR_swapchain
    VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR = 0x00000001,
    // Provided by VK_VERSION_1_1 with VK_KHR_swapchain
    VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR = 0x00000002,
    // Provided by VK_KHR_swapchain_mutable_format
    VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR = 0x00000004,
} VkSwapchainCreateFlagBitsKHR;
```

- **VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR** specifies that images created from the swapchain (i.e. with the **swapchain** member of **VkImageSwapchainCreateInfoKHR** set to this swapchain's handle) **must** use **VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT**. This flag is not supported in Vulkan SC [SCID-8].
- **VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR** specifies that images created from the swapchain are protected images.
- **VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR** specifies that the images of the swapchain **can** be used to create a **VkImageView** with a different format than what the swapchain was created with. The list of allowed image view formats is specified by adding a **VkImageFormatListCreateInfo** structure to the **pNext** chain of **VkSwapchainCreateInfoKHR**. In addition, this flag also specifies that the swapchain **can** be created with usage flags that are not supported for the format the swapchain is created with but are supported for at least one of the allowed image view formats.

```
// Provided by VK_KHR_swapchain
typedef VkFlags VkSwapchainCreateFlagsKHR;
```

VkSwapchainCreateFlagsKHR is a bitmask type for setting a mask of zero or more **VkSwapchainCreateFlagBitsKHR**.

If the **pNext** chain of **VkSwapchainCreateInfoKHR** includes a **VkDeviceGroupSwapchainCreateInfoKHR** structure, then that structure includes a set of device group present modes that the swapchain **can** be used with.

The **VkDeviceGroupSwapchainCreateInfoKHR** structure is defined as:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
typedef struct VkDeviceGroupSwapchainCreateInfoKHR {
    VkStructureType          sType;
    const void*              pNext;
    VkDeviceGroupPresentModeFlagsKHR modes;
} VkDeviceGroupSwapchainCreateInfoKHR;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **modes** is a bitfield of modes that the swapchain **can** be used with.

If this structure is not present, `modes` is considered to be `VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR`.

Valid Usage (Implicit)

- VUID-VkDeviceGroupSwapchainCreateInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_GROUP_SWAPCHAIN_CREATE_INFO_KHR`
- VUID-VkDeviceGroupSwapchainCreateInfoKHR-modes-parameter
`modes` **must** be a valid combination of `VkDeviceGroupPresentModeFlagBitsKHR` values
- VUID-VkDeviceGroupSwapchainCreateInfoKHR-modes-requiredbitmask
`modes` **must** not be `0`

To enable surface counters when creating a swapchain, add a `VkSwapchainCounterCreateInfoEXT` structure to the `pNext` chain of `VkSwapchainCreateInfoKHR`. `VkSwapchainCounterCreateInfoEXT` is defined as:

```
// Provided by VK_EXT_display_control
typedef struct VkSwapchainCounterCreateInfoEXT {
    VkStructureType      sType;
    const void*          pNext;
    VkSurfaceCounterFlagsEXT surfaceCounters;
} VkSwapchainCounterCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `surfaceCounters` is a bitmask of `VkSurfaceCounterFlagBitsEXT` specifying surface counters to enable for the swapchain.

Valid Usage

- VUID-VkSwapchainCounterCreateInfoEXT-surfaceCounters-01244
The bits in `surfaceCounters` **must** be supported by `VkSwapchainCreateInfoKHR::surface`, as reported by `vkGetPhysicalDeviceSurfaceCapabilities2EXT`

Valid Usage (Implicit)

- VUID-VkSwapchainCounterCreateInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SWAPCHAIN_COUNTER_CREATE_INFO_EXT`
- VUID-VkSwapchainCounterCreateInfoEXT-surfaceCounters-parameter
`surfaceCounters` **must** be a valid combination of `VkSurfaceCounterFlagBitsEXT` values

The requested counters become active when the first presentation command for the associated

swapchain is processed by the presentation engine. To query the value of an active counter, use:

```
// Provided by VK_EXT_display_control
VkResult vkGetSwapchainCounterEXT(
    VkDevice          device,
    VkSwapchainKHR    swapchain,
    VkSurfaceCounterFlagBitsEXT counter,
    uint64_t*         pCounterValue);
```

- **device** is the [VkDevice](#) associated with **swapchain**.
- **swapchain** is the swapchain from which to query the counter value.
- **counter** is a [VkSurfaceCounterFlagBitsEXT](#) value specifying the counter to query.
- **pCounterValue** will return the current value of the counter.

If a counter is not available because the swapchain is out of date, the implementation **may** return [VK_ERROR_OUT_OF_DATE_KHR](#).

If [VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations](#) is [VK_TRUE](#), [vkGetSwapchainCounterEXT](#) **must** not return [VK_ERROR_OUT_OF_HOST_MEMORY](#).

Valid Usage

- VUID-vkGetSwapchainCounterEXT-swapchain-01245
One or more present commands on **swapchain** **must** have been processed by the presentation engine

Valid Usage (Implicit)

- VUID-vkGetSwapchainCounterEXT-device-parameter
device **must** be a valid [VkDevice](#) handle
- VUID-vkGetSwapchainCounterEXT-swapchain-parameter
swapchain **must** be a valid [VkSwapchainKHR](#) handle
- VUID-vkGetSwapchainCounterEXT-counter-parameter
counter **must** be a valid [VkSurfaceCounterFlagBitsEXT](#) value
- VUID-vkGetSwapchainCounterEXT-pCounterValue-parameter
pCounterValue **must** be a valid pointer to a [uint64_t](#) value
- VUID-vkGetSwapchainCounterEXT-swapchain-parent
swapchain **must** have been created, allocated, or retrieved from **device**

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_OUT_OF_DATE_KHR`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

When the `VK_KHR_display_swapchain` extension is enabled, multiple swapchains that share presentable images are created by calling:

```
// Provided by VK_KHR_display_swapchain
VkResult vkCreateSharedSwapchainsKHR(
    VkDevice          device,
    uint32_t          swapchainCount,
    const VkSwapchainCreateInfoKHR* pCreateInfos,
    const VkAllocationCallbacks*   pAllocator,
    VkSwapchainKHR*    pSwapchains);
```

- `device` is the device to create the swapchains for.
- `swapchainCount` is the number of swapchains to create.
- `pCreateInfos` is a pointer to an array of `VkSwapchainCreateInfoKHR` structures specifying the parameters of the created swapchains.
- `pAllocator` is the allocator used for host memory allocated for the swapchain objects when there is no more specific allocator available (see [Memory Allocation](#)).
- `pSwapchains` is a pointer to an array of `VkSwapchainKHR` handles in which the created swapchain objects will be returned.

`vkCreateSharedSwapchainsKHR` is similar to `vkCreateSwapchainKHR`, except that it takes an array of `VkSwapchainCreateInfoKHR` structures, and returns an array of swapchain objects.

The swapchain creation parameters that affect the properties and number of presentable images **must** match between all the swapchains. If the displays used by any of the swapchains do not use the same presentable image layout or are incompatible in a way that prevents sharing images, swapchain creation will fail with the result code `VK_ERROR_INCOMPATIBLE_DISPLAY_KHR`. If any error occurs, no swapchains will be created. Images presented to multiple swapchains **must** be re-acquired from all of them before being modified. After destroying one or more of the swapchains, the remaining swapchains and the presentable images **can** continue to be used.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`,

`vkCreateSharedSwapchainsKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkCreateSharedSwapchainsKHR-device-05068

The number of swapchains currently allocated from `device` plus `swapchainCount` **must** be less than or equal to the total number of swapchains requested via `VkDeviceObjectReservationCreateInfo::swapchainRequestCount` specified when `device` was created

Valid Usage (Implicit)

- VUID-vkCreateSharedSwapchainsKHR-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkCreateSharedSwapchainsKHR-pCreateInfos-parameter
`pCreateInfos` **must** be a valid pointer to an array of `swapchainCount` valid `VkSwapchainCreateInfoKHR` structures
- VUID-vkCreateSharedSwapchainsKHR-pAllocator-null
`pAllocator` **must** be `NULL`
- VUID-vkCreateSharedSwapchainsKHR-pSwapchains-parameter
`pSwapchains` **must** be a valid pointer to an array of `swapchainCount` `VkSwapchainKHR` handles
- VUID-vkCreateSharedSwapchainsKHR-device-queuecount
The device **must** have been created with at least `1` queue
- VUID-vkCreateSharedSwapchainsKHR-swapchainCount-arraylength
`swapchainCount` **must** be greater than `0`

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_INCOMPATIBLE_DISPLAY_KHR`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_SURFACE_LOST_KHR`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

To obtain the array of presentable images associated with a swapchain, call:

```
// Provided by VK_KHR_swapchain
VkResult vkGetSwapchainImagesKHR(
    VkDevice          device,
    VkSwapchainKHR    swapchain,
    uint32_t*         pSwapchainImageCount,
    VkImage*          pSwapchainImages);
```

- **device** is the device associated with **swapchain**.
- **swapchain** is the swapchain to query.
- **pSwapchainImageCount** is a pointer to an integer related to the number of presentable images available or queried, as described below.
- **pSwapchainImages** is either **NULL** or a pointer to an array of **VkImage** handles.

If **pSwapchainImages** is **NULL**, then the number of presentable images for **swapchain** is returned in **pSwapchainImageCount**. Otherwise, **pSwapchainImageCount** **must** point to a variable set by the application to the number of elements in the **pSwapchainImages** array, and on return the variable is overwritten with the number of structures actually written to **pSwapchainImages**. If the value of **pSwapchainImageCount** is less than the number of presentable images for **swapchain**, at most **pSwapchainImageCount** structures will be written, and **VK_INCOMPLETE** will be returned instead of **VK_SUCCESS**, to indicate that not all the available presentable images were returned.

If **VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations** is **VK_TRUE**, **vkGetSwapchainImagesKHR** **must** not return **VK_ERROR_OUT_OF_HOST_MEMORY**.

Valid Usage (Implicit)

- VUID-vkGetSwapchainImagesKHR-device-parameter **device** **must** be a valid **VkDevice** handle
- VUID-vkGetSwapchainImagesKHR-swapchain-parameter **swapchain** **must** be a valid **VkSwapchainKHR** handle
- VUID-vkGetSwapchainImagesKHR-pSwapchainImageCount-parameter **pSwapchainImageCount** **must** be a valid pointer to a **uint32_t** value
- VUID-vkGetSwapchainImagesKHR-pSwapchainImages-parameter
If the value referenced by **pSwapchainImageCount** is not 0, and **pSwapchainImages** is not **NULL**, **pSwapchainImages** **must** be a valid pointer to an array of **pSwapchainImageCount** **VkImage** handles
- VUID-vkGetSwapchainImagesKHR-swapchain-parent **swapchain** **must** have been created, allocated, or retrieved from **device**

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

NOTE

By knowing all presentable images used in the swapchain, the application can create command buffers that reference these images prior to entering its main rendering loop.

Images returned by `vkGetSwapchainImagesKHR` are fully backed by memory before they are passed to the application, as if they are each bound completely and contiguously to a single `VkDeviceMemory` object. All presentable images are initially in the `VK_IMAGE_LAYOUT_UNDEFINED` layout, thus before using presentable images, the application **must** transition them to a valid layout for the intended use.

Images **can** also be created by using `vkCreateImage` with `VkImageSwapchainCreateInfoKHR` and bound to swapchain memory using `vkBindImageMemory2` with `VkBindImageMemorySwapchainInfoKHR`. These images **can** be used anywhere swapchain images are used, and are useful in logical devices with multiple physical devices to create peer memory bindings of swapchain memory. These images and bindings have no effect on what memory is presented. Unlike images retrieved from `vkGetSwapchainImagesKHR`, these images **must** be destroyed with `vkDestroyImage`.

To acquire an available presentable image to use, and retrieve the index of that image, call:

```
// Provided by VK_KHR_swapchain
VkResult vkAcquireNextImageKHR(
    VkDevice           device,
    VkSwapchainKHR     swapchain,
    uint64_t           timeout,
    VkSemaphore        semaphore,
    VkFence             fence,
    uint32_t*          pImageIndex);
```

- `device` is the device associated with `swapchain`.
- `swapchain` is the non-retired swapchain from which an image is being acquired.
- `timeout` specifies how long the function waits, in nanoseconds, if no image is available.

- **semaphore** is `VK_NULL_HANDLE` or a semaphore to signal.
- **fence** is `VK_NULL_HANDLE` or a fence to signal.
- **pImageIndex** is a pointer to a `uint32_t` in which the index of the next image to use (i.e. an index into the array of images returned by `vkGetSwapchainImagesKHR`) is returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkAcquireNextImageKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkAcquireNextImageKHR-swapchain-01285
swapchain **must** not be in the retired state
- VUID-vkAcquireNextImageKHR-semaphore-01286
If **semaphore** is not `VK_NULL_HANDLE`, it **must** be unsignaled
- VUID-vkAcquireNextImageKHR-semaphore-01779
If **semaphore** is not `VK_NULL_HANDLE`, it **must** not have any uncompleted signal or wait operations pending
- VUID-vkAcquireNextImageKHR-fence-01287
If **fence** is not `VK_NULL_HANDLE`, **fence** **must** be unsignaled
- VUID-vkAcquireNextImageKHR-fence-10066
If **fence** is not `VK_NULL_HANDLE`, **fence** **must** not be associated with any other queue command that has not yet completed execution on that queue
- VUID-vkAcquireNextImageKHR-semaphore-01780
semaphore and **fence** **must** not both be equal to `VK_NULL_HANDLE`
- VUID-vkAcquireNextImageKHR-surface-07783
If **forward progress** cannot be guaranteed for the **surface** used to create the **swapchain** member of **pAcquireInfo**, **timeout** **must** not be `UINT64_MAX`
- VUID-vkAcquireNextImageKHR-semaphore-03265
semaphore **must** have a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_BINARY`

Valid Usage (Implicit)

- VUID-vkAcquireNextImageKHR-device-parameter
device **must** be a valid `VkDevice` handle
- VUID-vkAcquireNextImageKHR-swapchain-parameter
swapchain **must** be a valid `VkSwapchainKHR` handle
- VUID-vkAcquireNextImageKHR-semaphore-parameter
If **semaphore** is not `VK_NULL_HANDLE`, **semaphore** **must** be a valid `VkSemaphore` handle
- VUID-vkAcquireNextImageKHR-fence-parameter
If **fence** is not `VK_NULL_HANDLE`, **fence** **must** be a valid `VkFence` handle
- VUID-vkAcquireNextImageKHR-pImageIndex-parameter

`pImageIndex` **must** be a valid pointer to a `uint32_t` value

- `VUID-vkAcquireNextImageKHR-swapchain-parent`
`swapchain` **must** have been created, allocated, or retrieved from `device`
- `VUID-vkAcquireNextImageKHR-semaphore-parent`
If `semaphore` is a valid handle, it **must** have been created, allocated, or retrieved from `device`
- `VUID-vkAcquireNextImageKHR-fence-parent`
If `fence` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `swapchain` **must** be externally synchronized
- Host access to `semaphore` **must** be externally synchronized
- Host access to `fence` **must** be externally synchronized

Return Codes

Success

- `VK_NOT_READY`
- `VK_SUBOPTIMAL_KHR`
- `VK_SUCCESS`
- `VK_TIMEOUT`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_OUT_OF_DATE_KHR`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_SURFACE_LOST_KHR`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

If an image is acquired successfully, `vkAcquireNextImageKHR` **must** either return `VK_SUCCESS` or `VK_SUBOPTIMAL_KHR`. The implementation **may** return `VK_SUBOPTIMAL_KHR` if the swapchain no longer matches the surface properties exactly, but **can** still be used for presentation.

When successful, `vkAcquireNextImageKHR` acquires a presentable image from `swapchain` that an application **can** use, and sets `pImageIndex` to the index of that image within the swapchain. The presentation engine **may** not have finished reading from the image at the time it is acquired, so the application **must** use `semaphore` and/or `fence` to ensure that the image layout and contents are not

modified until the presentation engine reads have completed. Once `vkAcquireNextImageKHR` successfully acquires an image, the semaphore signal operation referenced by `semaphore`, if not `VK_NULL_HANDLE`, and the fence signal operation referenced by `fence`, if not `VK_NULL_HANDLE`, are submitted for execution. If `vkAcquireNextImageKHR` does not successfully acquire an image, `semaphore` and `fence` are unaffected. The order in which images are acquired is implementation-dependent, and **may** be different than the order the images were presented.

If `timeout` is zero, then `vkAcquireNextImageKHR` does not wait, and will either successfully acquire an image, or fail and return `VK_NOT_READY` if no image is available.

If the specified timeout period expires before an image is acquired, `vkAcquireNextImageKHR` returns `VK_TIMEOUT`. If `timeout` is `UINT64_MAX`, the timeout period is treated as infinite, and `vkAcquireNextImageKHR` will block until an image is acquired or an error occurs.

Let S be the number of images in `swapchain`. Let M be the value of `VkSurfaceCapabilitiesKHR::minImageCount`.

`vkAcquireNextImageKHR` **should** not be called if the number of images that the application has currently acquired is greater than S-M. If `vkAcquireNextImageKHR` is called when the number of images that the application has currently acquired is less than or equal to S-M, `vkAcquireNextImageKHR` **must** return in finite time with an allowed `VkResult` code.

NOTE

Returning a result in finite time guarantees that the implementation cannot deadlock an application, or suspend its execution indefinitely with correct API usage. Acquiring too many images at once may block indefinitely, which is covered by valid usage when attempting to use `UINT64_MAX`. For example, a scenario here is when a compositor holds on to images which are currently being presented, and there are not any vacant images left to be acquired.

If the swapchain images no longer match native surface properties, either `VK_SUBOPTIMAL_KHR` or `VK_ERROR_OUT_OF_DATE_KHR` **must** be returned. If `VK_ERROR_OUT_OF_DATE_KHR` is returned, no image is acquired and attempts to present previously acquired images to the swapchain will also fail with `VK_ERROR_OUT_OF_DATE_KHR`. Applications need to create a new swapchain for the surface to continue presenting if `VK_ERROR_OUT_OF_DATE_KHR` is returned.

NOTE

`VK_SUBOPTIMAL_KHR` **may** happen, for example, if the platform surface has been resized but the platform is able to scale the presented images to the new size to produce valid surface updates. It is up to the application to decide whether it prefers to continue using the current swapchain in this state, or to re-create the swapchain to better match the platform surface properties.

If device loss occurs (see [Lost Device](#)) before the timeout has expired, `vkAcquireNextImageKHR` **must** return in finite time with either one of the allowed success codes, or `VK_ERROR_DEVICE_LOST`.

If `semaphore` is not `VK_NULL_HANDLE`, the semaphore **must** be unsignaled, with no signal or wait operations pending. It will become signaled when the application **can** use the image.

NOTE

Use of `semaphore` allows rendering operations to be recorded and submitted before

the presentation engine has completed its use of the image.

If **fence** is not equal to **VK_NULL_HANDLE**, the fence **must** be unsignaled, with no signal operations pending. It will become signaled when the application **can** use the image.

NOTE

Applications **should** not rely on **vkAcquireNextImageKHR** blocking in order to meter their rendering speed. The implementation **may** return from this function immediately regardless of how many presentation requests are queued, and regardless of when queued presentation requests will complete relative to the call. Instead, applications **can** use **fence** to meter their frame generation work to match the presentation rate.

An application **must** wait until either the **semaphore** or **fence** is signaled before accessing the image's data.

NOTE

When the presentable image will be accessed by some stage S, the recommended idiom for ensuring correct synchronization is:

- The **VkSubmitInfo** used to submit the image layout transition for execution includes **vkAcquireNextImageKHR::semaphore** in its **pWaitSemaphores** member, with the corresponding element of **pWaitDstStageMask** including S.
- The **synchronization command** that performs any necessary image layout transition includes S in both the **srcStageMask** and **dstStageMask**.

After a successful return, the image indicated by **pImageIndex** and its data will be unmodified compared to when it was presented.

NOTE

Exclusive ownership of presentable images corresponding to a swapchain created with **VK_SHARING_MODE_EXCLUSIVE** as defined in **Resource Sharing** is not altered by a call to **vkAcquireNextImageKHR**. That means upon the first acquisition from such a swapchain presentable images are not owned by any queue family, while at subsequent acquisitions the presentable images remain owned by the queue family the image was previously presented on.

The possible return values for **vkAcquireNextImageKHR** depend on the **timeout** provided:

- **VK_SUCCESS** is returned if an image became available.
- **VK_ERROR_SURFACE_LOST_KHR** is returned if the surface becomes no longer available.
- **VK_NOT_READY** is returned if **timeout** is zero and no image was available.
- **VK_TIMEOUT** is returned if **timeout** is greater than zero and less than **UINT64_MAX**, and no image became available within the time allowed.
- **VK_SUBOPTIMAL_KHR** is returned if an image became available, and the swapchain no longer matches the surface properties exactly, but **can** still be used to present to the surface successfully.

NOTE

This **may** happen, for example, if the platform surface has been resized but the

platform is able to scale the presented images to the new size to produce valid surface updates. It is up to the application to decide whether it prefers to continue using the current swapchain indefinitely or temporarily in this state, or to re-create the swapchain to better match the platform surface properties.

- `VK_ERROR_OUT_OF_DATE_KHR` is returned if the surface has changed in such a way that it is no longer compatible with the swapchain, and further presentation requests using the swapchain will fail. Applications **must** query the new surface properties and recreate their swapchain if they wish to continue presenting to the surface.

If the native surface and presented image sizes no longer match, presentation **may** fail. If presentation does succeed, the mapping from the presented image to the native surface is implementation-defined. It is the application's responsibility to detect surface size changes and react appropriately. If presentation fails because of a mismatch in the surface and presented image sizes, a `VK_ERROR_OUT_OF_DATE_KHR` error will be returned.

NOTE

For example, consider a 4x3 window/surface that gets resized to be 3x4 (taller than wider). On some window systems, the portion of the window/surface that was previously and still is visible (the 3x3 part) will contain the same contents as before, while the remaining parts of the window will have undefined contents. Other window systems **may** squash/stretch the image to fill the new window size without any undefined contents, or apply some other mapping.

To acquire an available presentable image to use, and retrieve the index of that image, call:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
VkResult vkAcquireNextImage2KHR(
    VkDevice device,
    const VkAcquireNextImageInfoKHR* pAcquireInfo,
    uint32_t* pImageIndex);
```

- `device` is the device associated with `swapchain`.
- `pAcquireInfo` is a pointer to a `VkAcquireNextImageInfoKHR` structure containing parameters of the acquire.
- `pImageIndex` is a pointer to a `uint32_t` value specifying the index of the next image to use.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkAcquireNextImage2KHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkAcquireNextImage2KHR-surface-07784

If **forward progress** cannot be guaranteed for the `surface` used to create `swapchain`, the `timeout` member of `pAcquireInfo` **must** not be `UINT64_MAX`

Valid Usage (Implicit)

- VUID-vkAcquireNextImage2KHR-device-parameter **device** must be a valid [VkDevice](#) handle
- VUID-vkAcquireNextImage2KHR-pAcquireInfo-parameter **pAcquireInfo** must be a valid pointer to a valid [VkAcquireNextImageInfoKHR](#) structure
- VUID-vkAcquireNextImage2KHR-pImageIndex-parameter **pImageIndex** must be a valid pointer to a [uint32_t](#) value

Return Codes

Success

- [VK_NOT_READY](#)
- [VK_SUBOPTIMAL_KHR](#)
- [VK_SUCCESS](#)
- [VK_TIMEOUT](#)

Failure

- [VK_ERROR_DEVICE_LOST](#)
- [VK_ERROR_OUT_OF_DATE_KHR](#)
- [VK_ERROR_OUT_OF_DEVICE_MEMORY](#)
- [VK_ERROR_OUT_OF_HOST_MEMORY](#)
- [VK_ERROR_SURFACE_LOST_KHR](#)
- [VK_ERROR_UNKNOWN](#)
- [VK_ERROR_VALIDATION_FAILED](#)

The [VkAcquireNextImageInfoKHR](#) structure is defined as:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
typedef struct VkAcquireNextImageInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkSwapchainKHR     swapchain;
    uint64_t           timeout;
    VkSemaphore         semaphore;
    VkFence             fence;
    uint32_t           deviceMask;
} VkAcquireNextImageInfoKHR;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is [NULL](#) or a pointer to a structure extending this structure.

- **swapchain** is a non-retired swapchain from which an image is acquired.
- **timeout** specifies how long the function waits, in nanoseconds, if no image is available.
- **semaphore** is **VK_NULL_HANDLE** or a semaphore to signal.
- **fence** is **VK_NULL_HANDLE** or a fence to signal.
- **deviceMask** is a mask of physical devices for which the swapchain image will be ready to use when the semaphore or fence is signaled.

If **vkAcquireNextImageKHR** is used, the device mask is considered to include all physical devices in the logical device.

NOTE

vkAcquireNextImage2KHR signals at most one semaphore, even if the application requests waiting for multiple physical devices to be ready via the **deviceMask**. However, only a single physical device **can** wait on that semaphore, since the semaphore becomes unsignaled when the wait succeeds. For other physical devices to wait for the image to be ready, it is necessary for the application to submit semaphore signal operation(s) to that first physical device to signal additional semaphore(s) after the wait succeeds, which the other physical device(s) **can** wait upon.

Valid Usage

- VUID-VkAcquireNextImageInfoKHR-swapchain-01675
swapchain **must** not be in the retired state
- VUID-VkAcquireNextImageInfoKHR-semaphore-01288
If **semaphore** is not **VK_NULL_HANDLE**, it **must** be unsignaled
- VUID-VkAcquireNextImageInfoKHR-semaphore-01781
If **semaphore** is not **VK_NULL_HANDLE**, it **must** not have any uncompleted signal or wait operations pending
- VUID-VkAcquireNextImageInfoKHR-fence-01289
If **fence** is not **VK_NULL_HANDLE**, **fence** **must** be unsignaled
- VUID-VkAcquireNextImageInfoKHR-fence-10067
If **fence** is not **VK_NULL_HANDLE**, **fence** **must** not be associated with any other queue command that has not yet completed execution on that queue
- VUID-VkAcquireNextImageInfoKHR-semaphore-01782
semaphore and **fence** **must** not both be equal to **VK_NULL_HANDLE**
- VUID-VkAcquireNextImageInfoKHR-deviceMask-01290
deviceMask **must** be a valid device mask
- VUID-VkAcquireNextImageInfoKHR-deviceMask-01291
deviceMask **must** not be zero
- VUID-VkAcquireNextImageInfoKHR-semaphore-03266
semaphore **must** have a **VkSemaphoreType** of **VK_SEMAPHORE_TYPE_BINARY**

Valid Usage (Implicit)

- VUID-VkAcquireNextImageInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_ACQUIRE_NEXT_IMAGE_INFO_KHR`
- VUID-VkAcquireNextImageInfoKHR-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkAcquireNextImageInfoKHR-swapchain-parameter
`swapchain` **must** be a valid `VkSwapchainKHR` handle
- VUID-VkAcquireNextImageInfoKHR-semaphore-parameter
If `semaphore` is not `VK_NULL_HANDLE`, `semaphore` **must** be a valid `VkSemaphore` handle
- VUID-VkAcquireNextImageInfoKHR-fence-parameter
If `fence` is not `VK_NULL_HANDLE`, `fence` **must** be a valid `VkFence` handle
- VUID-VkAcquireNextImageInfoKHR-commonparent
Each of `fence`, `semaphore`, and `swapchain` that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `swapchain` **must** be externally synchronized
- Host access to `semaphore` **must** be externally synchronized
- Host access to `fence` **must** be externally synchronized

After queueing all rendering commands and transitioning the image to the correct layout, to queue an image for presentation, call:

```
// Provided by VK_KHR_swapchain
VkResult vkQueuePresentKHR(
    VkQueue queue,
    const VkPresentInfoKHR* pPresentInfo);
```

- `queue` is a queue that is capable of presentation to the target surface's platform on the same device as the image's swapchain.
- `pPresentInfo` is a pointer to a `VkPresentInfoKHR` structure specifying parameters of the presentation.

NOTE

There is no requirement for an application to present images in the same order that they were acquired - applications can arbitrarily present any image that is currently acquired.

NOTE

The origin of the native orientation of the surface coordinate system is not specified in the Vulkan specification; it depends on the platform. For most platforms the

origin is by default upper-left, meaning the pixel of the presented `VkImage` at coordinates (0,0) would appear at the upper left pixel of the platform surface (assuming `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`, and the display standing the right way up).

The result codes `VK_ERROR_OUT_OF_DATE_KHR` and `VK_SUBOPTIMAL_KHR` have the same meaning when returned by `vkQueuePresentKHR` as they do when returned by `vkAcquireNextImageKHR`. If multiple swapchains are presented, the result code is determined by applying the following rules in order:

- If the device is lost, `VK_ERROR_DEVICE_LOST` is returned.
- If any of the target surfaces are no longer available the error `VK_ERROR_SURFACE_LOST_KHR` is returned.
- If any of the presents would have a result of `VK_ERROR_OUT_OF_DATE_KHR` if issued separately then `VK_ERROR_OUT_OF_DATE_KHR` is returned.
- If any of the presents would have a result of `VK_SUBOPTIMAL_KHR` if issued separately then `VK_SUBOPTIMAL_KHR` is returned.
- Otherwise `VK_SUCCESS` is returned.

Any writes to memory backing the images referenced by the `pImageIndices` and `pSwapchains` members of `pPresentInfo`, that are available before `vkQueuePresentKHR` is executed, are automatically made visible to the read access performed by the presentation engine. This automatic visibility operation for an image happens-after the semaphore signal operation, and happens-before the presentation engine accesses the image.

Presentation is a read-only operation that will not affect the content of the presentable images. Upon reacquiring the image and transitioning it away from the `VK_IMAGE_LAYOUT_PRESENT_SRC_KHR` layout, the contents will be the same as they were prior to transitioning the image to the present source layout and presenting it. However, if a mechanism other than Vulkan is used to modify the platform window associated with the swapchain, the content of all presentable images in the swapchain becomes undefined.

Calls to `vkQueuePresentKHR` **may** block, but **must** return in finite time. The processing of the presentation happens in issue order with other queue operations, but semaphores **must** be used to ensure that prior rendering and other commands in the specified queue complete before the presentation begins. The presentation command itself does not delay processing of subsequent commands on the queue. However, presentation requests sent to a particular queue are always performed in order. Exact presentation timing is controlled by the semantics of the presentation engine and native platform in use.

If an image is presented to a swapchain created from a display surface, the mode of the associated display will be updated, if necessary, to match the mode specified when creating the display surface. The mode switch and presentation of the specified image will be performed as one atomic operation.

Queueing an image for presentation defines a set of *queue operations*, including waiting on the semaphores and submitting a presentation request to the presentation engine. However, the scope of this set of queue operations does not include the actual processing of the image by the

presentation engine.

Regarding interaction with `vkQueueWaitIdle`, `vkQueuePresentKHR` behaves as if it signals a fence payload that is managed by the implementation and that has no associated `VkFence` handle. The fence will signal after all such queue operations have completed, and after the presentation engine has taken a reference to the payloads of all objects that the presentation engine accesses as part of the present operation. The fence **may** not wait for the present operation to complete.

If `vkQueuePresentKHR` fails to enqueue the corresponding set of queue operations, it **may** return `VK_ERROR_OUT_OF_HOST_MEMORY` or `VK_ERROR_OUT_OF_DEVICE_MEMORY`. If it does, the implementation **must** ensure that the state and contents of any resources or synchronization primitives referenced is unaffected by the call or its failure.

If `vkQueuePresentKHR` fails in such a way that the implementation is unable to make that guarantee, the implementation **must** return `VK_ERROR_DEVICE_LOST`.

However, if the presentation request is rejected by the presentation engine with an error `VK_ERROR_OUT_OF_DATE_KHR`, or `VK_ERROR_SURFACE_LOST_KHR`, the set of queue operations are still considered to be enqueued and thus any semaphore wait operation specified in `VkPresentInfoKHR` will execute when the corresponding queue operation is complete.

`vkQueuePresentKHR` releases the acquisition of the images referenced by `imageIndices`. The queue family corresponding to the queue `vkQueuePresentKHR` is executed on **must** have ownership of the presented images as defined in [Resource Sharing](#). `vkQueuePresentKHR` does not alter the queue family ownership, but the presented images **must** not be used again before they have been reacquired using `vkAcquireNextImageKHR`.

NOTE

The application **can** continue to present any acquired images from a retired swapchain as long as the swapchain has not entered a state that causes `vkQueuePresentKHR` to return `VK_ERROR_OUT_OF_DATE_KHR`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkQueuePresentKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkQueuePresentKHR-pSwapchains-01292
Each element of `pSwapchains` member of `pPresentInfo` **must** be a swapchain that is created for a surface for which presentation is supported from `queue` as determined using a call to `vkGetPhysicalDeviceSurfaceSupportKHR`
- VUID-vkQueuePresentKHR-pSwapchains-01293
If more than one member of `pSwapchains` was created from a display surface, all display surfaces referenced that refer to the same display **must** use the same display mode
- VUID-vkQueuePresentKHR-pWaitSemaphores-01294
When a semaphore wait operation referring to a binary semaphore defined by the elements of the `pWaitSemaphores` member of `pPresentInfo` executes on `queue`, there **must** be no other queues waiting on the same semaphore

- VUID-vkQueuePresentKHR-pWaitSemaphores-03267

All elements of the `pWaitSemaphores` member of `pPresentInfo` **must** be created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_BINARY`

- VUID-vkQueuePresentKHR-pWaitSemaphores-03268

All elements of the `pWaitSemaphores` member of `pPresentInfo` **must** reference a semaphore signal operation that has been submitted for execution and any `semaphore signal operations` on which it depends **must** have also been submitted for execution

Valid Usage (Implicit)

- VUID-vkQueuePresentKHR-queue-parameter
`queue` **must** be a valid `VkQueue` handle
- VUID-vkQueuePresentKHR-pPresentInfo-parameter
`pPresentInfo` **must** be a valid pointer to a valid `VkPresentInfoKHR` structure

Host Synchronization

- Host access to `queue` **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
-	-	Any	-

Return Codes

Success

- `VK_SUBOPTIMAL_KHR`
- `VK_SUCCESS`

Failure

- `VK_ERROR_DEVICE_LOST`
- `VK_ERROR_OUT_OF_DATE_KHR`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_SURFACE_LOST_KHR`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkPresentInfoKHR` structure is defined as:

```
// Provided by VK_KHR_swapchain
typedef struct VkPresentInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           waitSemaphoreCount;
    const VkSemaphore* pWaitSemaphores;
    uint32_t           swapchainCount;
    const VkSwapchainKHR* pSwapchains;
    const uint32_t*     pImageIndices;
    VkResult*          pResults;
} VkPresentInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `waitSemaphoreCount` is the number of semaphores to wait for before issuing the present request. The number **may** be zero.
- `pWaitSemaphores` is `NULL` or a pointer to an array of `VkSemaphore` objects with `waitSemaphoreCount` entries, and specifies the semaphores to wait for before issuing the present request.
- `swapchainCount` is the number of swapchains being presented to by this command.
- `pSwapchains` is a pointer to an array of `VkSwapchainKHR` objects with `swapchainCount` entries.
- `pImageIndices` is a pointer to an array of indices into the array of each swapchain's presentable images, with `swapchainCount` entries. Each entry in this array identifies the image to present on the corresponding entry in the `pSwapchains` array.
- `pResults` is a pointer to an array of `VkResult` typed elements with `swapchainCount` entries. Applications that do not need per-swapchain results **can** use `NULL` for `pResults`. If non-`NULL`, each entry in `pResults` will be set to the `VkResult` for presenting the swapchain corresponding to the same index in `pSwapchains`.

Before an application **can** present an image, the image's layout **must** be transitioned to the `VK_IMAGE_LAYOUT_PRESENT_SRC_KHR` layout, or for a shared presentable image the `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR` layout.

NOTE

When transitioning the image to the appropriate layout, there is no need to delay subsequent processing, or perform any visibility operations (as `vkQueuePresentKHR` performs automatic visibility operations). To achieve this, the `dstAccessMask` member of the `VkImageMemoryBarrier` **should** be `0`, and the `dstStageMask` parameter **should** be `VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT`.

Valid Usage

- VUID-VkPresentInfoKHR-pSwapchain-09231
Elements of `pSwapchain` **must** be unique

- VUID-VkPresentInfoKHR-pImageIndices-01430

Each element of **pImageIndices** **must** be the index of a presentable image acquired from the swapchain specified by the corresponding element of the **pSwapchains** array, and the presented image subresource **must** be in the **VK_IMAGE_LAYOUT_PRESENT_SRC_KHR** or **VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR** layout at the time the operation is executed on a **VkDevice**

Valid Usage (Implicit)

- VUID-VkPresentInfoKHR-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_PRESENT_INFO_KHR**
- VUID-VkPresentInfoKHR-pNext-pNext
Each **pNext** member of any structure (including this one) in the **pNext** chain **must** be either **NULL** or a pointer to a valid instance of **VkDeviceGroupPresentInfoKHR**, **VkDisplayPresentInfoKHR**, or **VkPresentRegionsKHR**
- VUID-VkPresentInfoKHR-sType-unique
The **sType** value of each structure in the **pNext** chain **must** be unique
- VUID-VkPresentInfoKHR-pWaitSemaphores-parameter
If **waitSemaphoreCount** is not 0, **pWaitSemaphores** **must** be a valid pointer to an array of **waitSemaphoreCount** valid **VkSemaphore** handles
- VUID-VkPresentInfoKHR-pSwapchains-parameter
pSwapchains **must** be a valid pointer to an array of **swapchainCount** valid **VkSwapchainKHR** handles
- VUID-VkPresentInfoKHR-pImageIndices-parameter
pImageIndices **must** be a valid pointer to an array of **swapchainCount** **uint32_t** values
- VUID-VkPresentInfoKHR-pResults-parameter
If **pResults** is not **NULL**, **pResults** **must** be a valid pointer to an array of **swapchainCount** **VkResult** values
- VUID-VkPresentInfoKHR-swapchainCount-arraylength
swapchainCount **must** be greater than 0
- VUID-VkPresentInfoKHR-commonparent
Both of the elements of **pSwapchains**, and the elements of **pWaitSemaphores** that are valid handles of non-ignored parameters **must** have been created, allocated, or retrieved from the same **VkDevice**

Host Synchronization

- Host access to each member of **pWaitSemaphores** **must** be externally synchronized
- Host access to each member of **pSwapchains** **must** be externally synchronized

When the **VK_KHR_incremental_present** extension is enabled, additional fields **can** be specified that

allow an application to specify that only certain rectangular regions of the presentable images of a swapchain are changed. This is an optimization hint that a presentation engine **may** use to only update the region of a surface that is actually changing. The application still **must** ensure that all pixels of a presented image contain the desired values, in case the presentation engine ignores this hint. An application **can** provide this hint by adding a `VkPresentRegionsKHR` structure to the `pNext` chain of the `VkPresentInfoKHR` structure.

The `VkPresentRegionsKHR` structure is defined as:

```
// Provided by VK_KHR_incremental_present
typedef struct VkPresentRegionsKHR {
    VkStructureType    sType;
    const void*        pNext;
    uint32_t           swapchainCount;
    const VkPresentRegionKHR* pRegions;
} VkPresentRegionsKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `swapchainCount` is the number of swapchains being presented to by this command.
- `pRegions` is `NULL` or a pointer to an array of `VkPresentRegionKHR` elements with `swapchainCount` entries. If not `NULL`, each element of `pRegions` contains the region that has changed since the last present to the swapchain in the corresponding entry in the `VkPresentInfoKHR::pSwapchains` array.

Valid Usage

- VUID-VkPresentRegionsKHR-swapchainCount-01260
`swapchainCount` **must** be the same value as `VkPresentInfoKHR::swapchainCount`, where `VkPresentInfoKHR` is included in the `pNext` chain of this `VkPresentRegionsKHR` structure

Valid Usage (Implicit)

- VUID-VkPresentRegionsKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PRESENT_REGIONS_KHR`
- VUID-VkPresentRegionsKHR-pRegions-parameter
If `pRegions` is not `NULL`, `pRegions` **must** be a valid pointer to an array of `swapchainCount` valid `VkPresentRegionKHR` structures
- VUID-VkPresentRegionsKHR-swapchainCount-arraylength
`swapchainCount` **must** be greater than 0

For a given image and swapchain, the region to present is specified by the `VkPresentRegionKHR` structure, which is defined as:

```
// Provided by VK_KHR_incremental_present
typedef struct VkPresentRegionKHR {
    uint32_t      rectangleCount;
    const VkRectLayerKHR* pRectangles;
} VkPresentRegionKHR;
```

- **rectangleCount** is the number of rectangles in **pRectangles**, or zero if the entire image has changed and should be presented.
- **pRectangles** is either **NULL** or a pointer to an array of **VkRectLayerKHR** structures. The **VkRectLayerKHR** structure is the framebuffer coordinates, plus layer, of a portion of a presentable image that has changed and **must** be presented. If non-**NULL**, each entry in **pRectangles** is a rectangle of the given image that has changed since the last image was presented to the given swapchain. The rectangles **must** be specified relative to **VkSurfaceCapabilitiesKHR::currentTransform**, regardless of the swapchain's **preTransform**. The presentation engine will apply the **preTransform** transformation to the rectangles, along with any further transformation it applies to the image content.

Valid Usage (Implicit)

- VUID-VkPresentRegionKHR-pRectangles-parameter
If **rectangleCount** is not 0, and **pRectangles** is not **NULL**, **pRectangles** **must** be a valid pointer to an array of **rectangleCount** valid **VkRectLayerKHR** structures

The **VkRectLayerKHR** structure is defined as:

```
// Provided by VK_KHR_incremental_present
typedef struct VkRectLayerKHR {
    VkOffset2D    offset;
    VkExtent2D    extent;
    uint32_t      layer;
} VkRectLayerKHR;
```

- **offset** is the origin of the rectangle, in pixels.
- **extent** is the size of the rectangle, in pixels.
- **layer** is the layer of the image. For images with only one layer, the value of **layer** **must** be 0.

Some platforms allow the size of a surface to change, and then scale the pixels of the image to fit the surface. **VkRectLayerKHR** specifies pixels of the swapchain's image(s), which will be constant for the life of the swapchain.

Valid Usage

- VUID-VkRectLayerKHR-offset-04864
The sum of **offset** and **extent**, after being transformed according to the **preTransform**

member of the [VkSwapchainCreateInfoKHR](#) structure, **must** be no greater than the [imageExtent](#) member of the [VkSwapchainCreateInfoKHR](#) structure passed to [vkCreateSwapchainKHR](#)

- VUID-VkRectLayerKHR-layer-01262

[layer](#) **must** be less than the [imageArrayLayers](#) member of the [VkSwapchainCreateInfoKHR](#) structure passed to [vkCreateSwapchainKHR](#)

When the [VK_KHR_display_swapchain](#) extension is enabled, additional fields **can** be specified when presenting an image to a swapchain by setting [VkPresentInfoKHR::pNext](#) to point to a [VkDisplayPresentInfoKHR](#) structure.

The [VkDisplayPresentInfoKHR](#) structure is defined as:

```
// Provided by VK_KHR_display_swapchain
typedef struct VkDisplayPresentInfoKHR {
    VkStructureType    sType;
    const void*        pNext;
    VkRect2D           srcRect;
    VkRect2D           dstRect;
    VkBool32           persistent;
} VkDisplayPresentInfoKHR;
```

- [sType](#) is a [VkStructureType](#) value identifying this structure.
- [pNext](#) is [NULL](#) or a pointer to a structure extending this structure.
- [srcRect](#) is a rectangular region of pixels to present. It **must** be a subset of the image being presented. If [VkDisplayPresentInfoKHR](#) is not specified, this region will be assumed to be the entire presentable image.
- [dstRect](#) is a rectangular region within the visible region of the swapchain's display mode. If [VkDisplayPresentInfoKHR](#) is not specified, this region will be assumed to be the entire visible region of the swapchain's mode. If the specified rectangle is a subset of the display mode's visible region, content from display planes below the swapchain's plane will be visible outside the rectangle. If there are no planes below the swapchain's, the area outside the specified rectangle will be black. If portions of the specified rectangle are outside of the display's visible region, pixels mapping only to those portions of the rectangle will be discarded.
- [persistent](#): If this is [VK_TRUE](#), the display engine will enable buffered mode on displays that support it. This allows the display engine to stop sending content to the display until a new image is presented. The display will instead maintain a copy of the last presented image. This allows less power to be used, but **may** increase presentation latency. If [VkDisplayPresentInfoKHR](#) is not specified, persistent mode will not be used.

If the extent of the [srcRect](#) and [dstRect](#) are not equal, the presented pixels will be scaled accordingly.

Valid Usage

- VUID-VkDisplayPresentInfoKHR-srcRect-01257
`srcRect` **must** specify a rectangular region that is a subset of the image being presented
- VUID-VkDisplayPresentInfoKHR-dstRect-01258
`dstRect` **must** specify a rectangular region that is a subset of the `visibleRegion` parameter of the display mode the swapchain being presented uses
- VUID-VkDisplayPresentInfoKHR-persistentContent-01259
If the `persistentContent` member of the `VkDisplayPropertiesKHR` structure returned by `vkGetPhysicalDeviceDisplayPropertiesKHR` for the display the present operation targets is `VK_FALSE`, then `persistent` **must** be `VK_FALSE`

Valid Usage (Implicit)

- VUID-VkDisplayPresentInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DISPLAY_PRESENT_INFO_KHR`

If the `pNext` chain of `VkPresentInfoKHR` includes a `VkDeviceGroupPresentInfoKHR` structure, then that structure includes an array of device masks and a device group present mode.

The `VkDeviceGroupPresentInfoKHR` structure is defined as:

```
// Provided by VK_VERSION_1_1 with VK_KHR_swapchain
typedef struct VkDeviceGroupPresentInfoKHR {
    VkStructureType             sType;
    const void*                 pNext;
    uint32_t                    swapchainCount;
    const uint32_t*              pDeviceMasks;
    VkDeviceGroupPresentModeFlagsKHR mode;
} VkDeviceGroupPresentInfoKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `swapchainCount` is zero or the number of elements in `pDeviceMasks`.
- `pDeviceMasks` is a pointer to an array of device masks, one for each element of `VkPresentInfoKHR::pSwapchains`.
- `mode` is a `VkDeviceGroupPresentModeFlagsKHR` value specifying the device group present mode that will be used for this present.

If `mode` is `VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR`, then each element of `pDeviceMasks` selects which instance of the swapchain image is presented. Each element of `pDeviceMasks` **must** have exactly one bit set, and the corresponding physical device **must** have a presentation engine as reported by `VkDeviceGroupPresentCapabilitiesKHR`.

If `mode` is `VK_DEVICE_GROUP_PRESENT_MODE_REMOTE_BIT_KHR`, then each element of `pDeviceMasks` selects which instance of the swapchain image is presented. Each element of `pDeviceMasks` **must** have exactly one bit set, and some physical device in the logical device **must** include that bit in its `VkDeviceGroupPresentCapabilitiesKHR::presentMask`.

If `mode` is `VK_DEVICE_GROUP_PRESENT_MODE_SUM_BIT_KHR`, then each element of `pDeviceMasks` selects which instances of the swapchain image are component-wise summed and the sum of those images is presented. If the sum in any component is outside the representable range, the value of that component is undefined. Each element of `pDeviceMasks` **must** have a value for which all set bits are set in one of the elements of `VkDeviceGroupPresentCapabilitiesKHR::presentMask`.

If `mode` is `VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_MULTI_DEVICE_BIT_KHR`, then each element of `pDeviceMasks` selects which instance(s) of the swapchain images are presented. For each bit set in each element of `pDeviceMasks`, the corresponding physical device **must** have a presentation engine as reported by `VkDeviceGroupPresentCapabilitiesKHR`.

If `VkDeviceGroupPresentInfoKHR` is not provided or `swapchainCount` is zero then the masks are considered to be 1. If `VkDeviceGroupPresentInfoKHR` is not provided, `mode` is considered to be `VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR`.

Valid Usage

- VUID-VkDeviceGroupPresentInfoKHR-swapchainCount-01297
`swapchainCount` **must** equal 0 or `VkPresentInfoKHR::swapchainCount`
- VUID-VkDeviceGroupPresentInfoKHR-mode-01298
If `mode` is `VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_BIT_KHR`, then each element of `pDeviceMasks` **must** have exactly one bit set, and the corresponding element of `VkDeviceGroupPresentCapabilitiesKHR::presentMask` **must** be non-zero
- VUID-VkDeviceGroupPresentInfoKHR-mode-01299
If `mode` is `VK_DEVICE_GROUP_PRESENT_MODE_REMOTE_BIT_KHR`, then each element of `pDeviceMasks` **must** have exactly one bit set, and some physical device in the logical device **must** include that bit in its `VkDeviceGroupPresentCapabilitiesKHR::presentMask`
- VUID-VkDeviceGroupPresentInfoKHR-mode-01300
If `mode` is `VK_DEVICE_GROUP_PRESENT_MODE_SUM_BIT_KHR`, then each element of `pDeviceMasks` **must** have a value for which all set bits are set in one of the elements of `VkDeviceGroupPresentCapabilitiesKHR::presentMask`
- VUID-VkDeviceGroupPresentInfoKHR-mode-01301
If `mode` is `VK_DEVICE_GROUP_PRESENT_MODE_LOCAL_MULTI_DEVICE_BIT_KHR`, then for each bit set in each element of `pDeviceMasks`, the corresponding element of `VkDeviceGroupPresentCapabilitiesKHR::presentMask` **must** be non-zero
- VUID-VkDeviceGroupPresentInfoKHR-pDeviceMasks-01302
The value of each element of `pDeviceMasks` **must** be equal to the device mask passed in `VkAcquireNextImageInfoKHR::deviceMask` when the image index was last acquired
- VUID-VkDeviceGroupPresentInfoKHR-mode-01303
`mode` **must** have exactly one bit set, and that bit **must** have been included in `VkDeviceGroupSwapchainCreateInfoKHR::modes`

Valid Usage (Implicit)

- VUID-VkDeviceGroupPresentInfoKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_INFO_KHR`
- VUID-VkDeviceGroupPresentInfoKHR-pDeviceMasks-parameter
If `swapchainCount` is not 0, `pDeviceMasks` **must** be a valid pointer to an array of `swapchainCount` `uint32_t` values
- VUID-VkDeviceGroupPresentInfoKHR-mode-parameter
`mode` **must** be a valid `VkDeviceGroupPresentModeFlagBitsKHR` value

31.8. HDR Metadata

This section describes how to improve color reproduction of content to better reproduce colors as seen on the display used to originally optimize the content.

To provide HDR metadata to an implementation, call:

```
// Provided by VK_EXT_hdr_metadata
void vkSetHdrMetadataEXT(
    VkDevice          device,
    uint32_t          swapchainCount,
    const VkSwapchainKHR* pSwapchains,
    const VkHdrMetadataEXT* pMetadata);
```

- `device` is the logical device where the swapchain(s) were created.
- `swapchainCount` is the number of swapchains included in `pSwapchains`.
- `pSwapchains` is a pointer to an array of `swapchainCount` `VkSwapchainKHR` handles.
- `pMetadata` is a pointer to an array of `swapchainCount` `VkHdrMetadataEXT` structures.

The metadata will be applied to the specified `VkSwapchainKHR` objects at the next `vkQueuePresentKHR` call using that `VkSwapchainKHR` object. The metadata will persist until a subsequent `vkSetHdrMetadataEXT` changes it.

Valid Usage (Implicit)

- VUID-vkSetHdrMetadataEXT-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkSetHdrMetadataEXT-pSwapchains-parameter
`pSwapchains` **must** be a valid pointer to an array of `swapchainCount` valid `VkSwapchainKHR` handles
- VUID-vkSetHdrMetadataEXT-pMetadata-parameter
`pMetadata` **must** be a valid pointer to an array of `swapchainCount` valid `VkHdrMetadataEXT` structures

- VUID-vkSetHdrMetadataEXT-swapchainCount-arraylength
`swapchainCount` **must** be greater than 0
- VUID-vkSetHdrMetadataEXT-pSwapchains-parent
Each element of `pSwapchains` **must** have been created, allocated, or retrieved from `device`

The `VkHdrMetadataEXT` structure is defined as:

```
// Provided by VK_EXT_hdr_metadata
typedef struct VkHdrMetadataEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkXYColorEXT       displayPrimaryRed;
    VkXYColorEXT       displayPrimaryGreen;
    VkXYColorEXT       displayPrimaryBlue;
    VkXYColorEXT       whitePoint;
    float              maxLuminance;
    float              minLuminance;
    float              maxContentLightLevel;
    float              maxFrameAverageLightLevel;
} VkHdrMetadataEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `displayPrimaryRed` is a `VkXYColorEXT` structure specifying the red primary of the display used to optimize the content
- `displayPrimaryGreen` is a `VkXYColorEXT` structure specifying the green primary of the display used to optimize the content
- `displayPrimaryBlue` is a `VkXYColorEXT` structure specifying the blue primary of the display used to optimize the content
- `whitePoint` is a `VkXYColorEXT` structure specifying the white-point of the display used to optimize the content
- `maxLuminance` is the maximum luminance of the display used to optimize the content in nits
- `minLuminance` is the minimum luminance of the display used to optimize the content in nits
- `maxContentLightLevel` is the value in nits of the desired luminance for the brightest pixels in the displayed image.
- `maxFrameAverageLightLevel` is the value in nits of the average luminance of the frame which has the brightest average luminance anywhere in the content.

If any of the above values are unknown, they **can** be set to 0.

NOTE	The meta-data provided here is intended to be used as defined in the SMPTE 2086, CTA 861.3 and CIE 15:2004 specifications. The validity and use of this data is outside the scope of Vulkan.

Valid Usage (Implicit)

- VUID-VkHdrMetadataEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_HDR_METADATA_EXT`
- VUID-VkHdrMetadataEXT-pNext-pNext
`pNext` **must** be `NULL`

The `VkXYColorEXT` structure is defined as:

```
// Provided by VK_EXT_hdr_metadata
typedef struct VkXYColorEXT {
    float    x;
    float    y;
} VkXYColorEXT;
```

- `x` is the x chromaticity coordinate.
- `y` is the y chromaticity coordinate.

Chromaticity coordinates are as specified in CIE 15:2004 “Calculation of chromaticity coordinates” (Section 7.3) and are limited to between 0 and 1 for real colors.

Chapter 32. Extending Vulkan

New functionality **may** be added to Vulkan via either new extensions or new versions of the core, or new versions of an extension in some cases.

This chapter describes how Vulkan is versioned, how compatibility is affected between different versions, and compatibility rules that are followed by the Vulkan Working Group.

32.1. Functionality Levels

Functionality in Vulkan is divided into several different levels; global, instance-level, physical-device-level, and device-level.

- [VkInstance](#) and any objects created from a [VkInstance](#) other than [VkPhysicalDevice](#) are instance-level.
- [VkPhysicalDevice](#) is the only physical-device-level object.
- [VkDevice](#) and any objects created from a [VkDevice](#) are device-level.

The level of a command is the same as the level of its first parameter - if the first parameter is not a [dispatchable handle](#), it is a global command. Different levels of functionality **may** be advertised in different ways.

32.2. Instance and Device Versions

Starting with Vulkan 1.1, there are separate versions advertised for the Vulkan instance, and for each device supported on the system. This allows a system with multiple devices to advertise all devices at their full capabilities, even if those devices do not support the same version of Vulkan.

The instance version indicates which global and instance-level functionality is supported, while each device version indicates the physical-device-level and device-level functionality supported.

The instance version **can** be queried by calling [vkEnumerateInstanceVersion](#). Querying for this function via [vkGetInstanceProcAddr](#) will return [NULL](#) on implementations that only support Vulkan 1.0 functionality.

The device version **can** be queried by calling [vkGetPhysicalDeviceProperties](#) or [vkGetPhysicalDeviceProperties2](#), and is returned in [VkPhysicalDeviceProperties::apiVersion](#), encoded as described in [Version Numbers](#).

32.3. Core Versions

The Vulkan Specification is regularly updated with bug fixes and clarifications. Occasionally new functionality is added to the core and at some point it is expected that there will be a desire to perform a large, breaking change to the API. In order to indicate to developers how and when these changes are made to the specification, and to provide a way to identify each set of changes, the Vulkan API maintains a version number.

32.3.1. Version Numbers

The Vulkan version number comprises four parts indicating the variant, major, minor and patch version of the Vulkan API Specification.

The *variant* indicates the variant of the Vulkan API supported by the implementation. This is always 1 for the Vulkan SC API. The Base Vulkan API is variant 0.

NOTE

A non-zero variant indicates the API is a variant of the Vulkan API and applications will typically need to be modified to run against it. The variant field was a later addition to the version number, added in version 1.2.175 of the Base Vulkan Specification.

The *major version* indicates a significant change in the API, which will encompass a wholly new version of the specification.

The *minor version* indicates the incorporation of new functionality into the core specification.

The *patch version* indicates bug fixes, clarifications, and language improvements have been incorporated into the specification.

Compatibility guarantees made about versions of the API sharing any of the same version numbers are documented in [Core Versions](#)

The version number is used in several places in the API. In each such use, the version numbers are packed into a 32-bit integer as follows:

- The variant is a 3-bit integer packed into bits 31-29.
- The major version is a 7-bit integer packed into bits 28-22.
- The minor version number is a 10-bit integer packed into bits 21-12.
- The patch version number is a 12-bit integer packed into bits 11-0.

VK_API_VERSION_VARIANT extracts the API variant number from a packed version number:

```
// Provided by VK_VERSION_1_0
#define VK_API_VERSION_VARIANT(version) ((uint32_t)(version) >> 29U)
```

VK_API_VERSION_MAJOR extracts the API major version number from a packed version number:

```
// Provided by VK_VERSION_1_0
#define VK_API_VERSION_MAJOR(version) (((uint32_t)(version) >> 22U) & 0x7FU)
```

VK_API_VERSION_MINOR extracts the API minor version number from a packed version number:

```
// Provided by VK_VERSION_1_0
#define VK_API_VERSION_MINOR(version) (((uint32_t)(version) >> 12U) & 0x3FFU)
```

VK_API_VERSION_PATCH extracts the API patch version number from a packed version number:

```
// Provided by VK_VERSION_1_0
#define VK_API_VERSION_PATCH(version) (((uint32_t)(version) & 0xFFU)
```

VK_MAKE_API_VERSION constructs an API version number.

```
// Provided by VK_VERSION_1_0
#define VK_MAKE_API_VERSION(variant, major, minor, patch) \
    (((uint32_t)(variant)) << 29U) | (((uint32_t)(major)) << 22U) | \
    (((uint32_t)(minor)) << 12U) | ((uint32_t)(patch)))
```

- **variant** is the variant number.
- **major** is the major version number.
- **minor** is the minor version number.
- **patch** is the patch version number.

VK_API_VERSION_1_0 returns the API version number for Vulkan 1.0.0.

```
// Provided by VK_VERSION_1_0
// Vulkan 1.0 version number
#define VK_API_VERSION_1_0 VK_MAKE_API_VERSION(0, 1, 0, 0)// Patch version should always be set to 0
```

VK_API_VERSION_1_1 returns the API version number for Vulkan 1.1.0.

```
// Provided by VK_VERSION_1_1
// Vulkan 1.1 version number
#define VK_API_VERSION_1_1 VK_MAKE_API_VERSION(0, 1, 1, 0)// Patch version should always be set to 0
```

VK_API_VERSION_1_2 returns the API version number for Vulkan 1.2.0.

```
// Provided by VK_VERSION_1_2
// Vulkan 1.2 version number
#define VK_API_VERSION_1_2 VK_MAKE_API_VERSION(0, 1, 2, 0)// Patch version should always be set to 0
```

VKSC_API_VARIANT returns the API variant number for Vulkan SC.

```
// Provided by VKSC_VERSION_1_0
// Vulkan SC variant number
#define VKSC_API_VARIANT 1
```

`VKSC_API_VERSION_1_0` returns the API version number for Vulkan SC 1.0.0.

```
// Provided by VKSC_VERSION_1_0
// Vulkan SC 1.0 version number
#define VKSC_API_VERSION_1_0 VK_MAKE_API_VERSION(VKSC_API_VARIANT, 1, 0, 0) // Patch
version should always be set to 0
```

32.4. Layers

When a layer is enabled, it inserts itself into the call chain for Vulkan commands the layer is interested in. Layers **can** be used for a variety of tasks that extend the base behavior of Vulkan beyond what is required by the specification - such as call logging, tracing, validation, or providing additional extensions.

NOTE

For example, an implementation is not expected to check that the value of enums used by the application fall within allowed ranges. Instead, a validation layer would do those checks and flag issues. This avoids a performance penalty during production use of the application because those layers would not be enabled in production.

NOTE

Vulkan layers **may** wrap object handles (i.e. return a different handle value to the application than that generated by the implementation). This is generally discouraged, as it increases the probability of incompatibilities with new extensions. The validation layers wrap handles in order to track the proper use and destruction of each object. See the [“Architecture of the Vulkan Loader Interfaces”](#) document for additional information.

To query the available layers, call:

```
// Provided by VK_VERSION_1_0
VkResult vkEnumerateInstanceLayerProperties(
    uint32_t*                pPropertyCount,
    VkLayerProperties*        pProperties);
```

- `pPropertyCount` is a pointer to an integer related to the number of layer properties available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of `VkLayerProperties` structures.

If `pProperties` is `NULL`, then the number of layer properties available is returned in `pPropertyCount`. Otherwise, `pPropertyCount` **must** point to a variable set by the application to the number of elements

in the `pProperties` array, and on return the variable is overwritten with the number of structures actually written to `pProperties`. If `pPropertyCount` is less than the number of layer properties available, at most `pPropertyCount` structures will be written, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available properties were returned.

The list of available layers may change at any time due to actions outside of the Vulkan implementation, so two calls to `vkEnumerateInstanceLayerProperties` with the same parameters **may** return different results, or retrieve different `pPropertyCount` values or `pProperties` contents. Once an instance has been created, the layers enabled for that instance will continue to be enabled and valid for the lifetime of that instance, even if some of them become unavailable for future instances.

Valid Usage (Implicit)

- VUID-vkEnumerateInstanceLayerProperties-pPropertyCount-parameter
`pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkEnumerateInstanceLayerProperties-pProperties-parameter
If the value referenced by `pPropertyCount` is not 0, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` `VkLayerProperties` structures

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkLayerProperties` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkLayerProperties {
    char        layerName[VK_MAX_EXTENSION_NAME_SIZE];
    uint32_t    specVersion;
    uint32_t    implementationVersion;
    char        description[VK_MAX_DESCRIPTION_SIZE];
} VkLayerProperties;
```

- `layerName` is an array of `VK_MAX_EXTENSION_NAME_SIZE` `char` containing a null-terminated UTF-8 string which is the name of the layer. Use this name in the `ppEnabledLayerNames` array passed in

the [VkInstanceCreateInfo](#) structure to enable this layer for an instance.

- **specVersion** is the Vulkan version the layer was written to, encoded as described in [Version Numbers](#).
- **implementationVersion** is the version of this layer. It is an integer, increasing with backward compatible changes.
- **description** is an array of **VK_MAX_DESCRIPTION_SIZE** **char** containing a null-terminated UTF-8 string which provides additional details that **can** be used by the application to identify the layer.

VK_MAX_EXTENSION_NAME_SIZE is the length in **char** values of an array containing a layer or extension name string, as returned in [VkLayerProperties::layerName](#), [VkExtensionProperties::extensionName](#), and other queries.

```
#define VK_MAX_EXTENSION_NAME_SIZE    256U
```

VK_MAX_DESCRIPTION_SIZE is the length in **char** values of an array containing a string with additional descriptive information about a query, as returned in [VkLayerProperties::description](#) and other queries.

```
#define VK_MAX_DESCRIPTION_SIZE    256U
```

To enable a layer, the name of the layer **should** be added to the **ppEnabledLayerNames** member of [VkInstanceCreateInfo](#) when creating a [VkInstance](#).

Loader implementations **may** provide mechanisms outside the Vulkan API for enabling specific layers. Layers enabled through such a mechanism are *implicitly enabled*, while layers enabled by including the layer name in the **ppEnabledLayerNames** member of [VkInstanceCreateInfo](#) are *explicitly enabled*. Implicitly enabled layers are loaded before explicitly enabled layers, such that implicitly enabled layers are closer to the application, and explicitly enabled layers are closer to the driver. Except where otherwise specified, implicitly enabled and explicitly enabled layers differ only in the way they are enabled, and the order in which they are loaded. Explicitly enabling a layer that is implicitly enabled results in this layer being loaded as an implicitly enabled layer; it has no additional effect.

To enumerate device layers, call:

WARNING

This functionality is superseded by [Vulkan Version 1.0](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
VkResult vkEnumerateDeviceLayerProperties(
    VkPhysicalDevice    physicalDevice,
    uint32_t*           pPropertyCount,
    VkLayerProperties*   pProperties);
```

- **physicalDevice** is the physical device that will be queried.

- `pPropertyCount` is a pointer to an integer related to the number of layer properties available or queried.
- `pProperties` is either `NULL` or a pointer to an array of `VkLayerProperties` structures.

Physical device layers are not supported. `pPropertyCount` will be 0, and `VK_SUCCESS` is returned.

Valid Usage (Implicit)

- `VUID-vkEnumerateDeviceLayerProperties-physicalDevice-parameter`
`physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- `VUID-vkEnumerateDeviceLayerProperties-pPropertyCount-parameter`
`pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- `VUID-vkEnumerateDeviceLayerProperties-pProperties-parameter`
If the value referenced by `pPropertyCount` is not 0, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` `VkLayerProperties` structures

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `ppEnabledLayerNames` and `enabledLayerCount` members of `VkDeviceCreateInfo` are legacy and their values **must** be ignored by implementations. The sequence of layers active for a device will be exactly the sequence of layers enabled when the parent instance was created.

32.5. Extensions

Extensions **may** define new Vulkan commands, structures, and enumerants. For compilation purposes, the interfaces defined by registered extensions, including new structures and enumerants as well as function pointer types for new commands, are defined in the Khronos-supplied `vulkan_sc_core.h` together with the core API. However, commands defined by extensions **may** not be available for static linking - in which case function pointers to these commands **should** be queried at runtime as described in [Command Function Pointers](#). Extensions **may** be provided by layers as well as by a Vulkan implementation.

Because extensions **may** extend or change the behavior of the Vulkan API, extension authors **should** add support for their extensions to the Khronos validation layers. This is especially important for new commands whose parameters have been wrapped by the validation layers. See the [“Architecture of the Vulkan Loader Interfaces”](#) document for additional information.

NOTE

To enable an instance extension, the name of the extension **can** be added to the

`ppEnabledExtensionNames` member of `VkInstanceCreateInfo` when creating a `VkInstance`.

To enable a device extension, the name of the extension **can** be added to the `ppEnabledExtensionNames` member of `VkDeviceCreateInfo` when creating a `VkDevice`.

Physical-Device-Level functionality does not have any enabling mechanism and **can** be used as long as the `VkPhysicalDevice` supports the device extension as determined by `vkEnumerateDeviceExtensionProperties`.

Enabling an extension (with no further use of that extension) does not change the behavior of functionality exposed by the core Vulkan API or any other extension, other than making valid the use of the commands, enums and structures defined by that extension.

Valid Usage sections for individual commands and structures do not currently contain which extensions have to be enabled in order to make their use valid, although they might do so in the future. It is defined only in the [Valid Usage for Extensions](#) section.

32.5.1. Instance Extensions

Instance extensions add new [global or instance-level functionality](#) to the API, outside of the core specification. Instance extensions **may** also add [physical-device-level functionality](#).

To query the available instance extensions, call:

```
// Provided by VK_VERSION_1_0
VkResult vkEnumerateInstanceExtensionProperties(
    const char*           pLayerName,
    uint32_t*             pPropertyCount,
    VkExtensionProperties* pProperties);
```

- `pLayerName` is either `NULL` or a pointer to a null-terminated UTF-8 string naming the layer to retrieve extensions from.
- `pPropertyCount` is a pointer to an integer related to the number of extension properties available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of `VkExtensionProperties` structures.

When `pLayerName` parameter is `NULL`, only extensions provided by the Vulkan implementation or by implicitly enabled layers are returned. When `pLayerName` is the name of a layer, the instance extensions provided by that layer are returned.

If `pProperties` is `NULL`, then the number of extensions properties available is returned in `pPropertyCount`. Otherwise, `pPropertyCount` **must** point to a variable set by the application to the number of elements in the `pProperties` array, and on return the variable is overwritten with the number of structures actually written to `pProperties`. If `pPropertyCount` is less than the number of extension properties available, at most `pPropertyCount` structures will be written, and `VK_INCOMPLETE`

will be returned instead of `VK_SUCCESS`, to indicate that not all the available properties were returned.

Because the list of available layers may change externally between calls to `vkEnumerateInstanceExtensionProperties`, two calls may retrieve different results if a `pLayerName` is available in one call but not in another. The extensions supported by a layer may also change between two calls, e.g. if the layer implementation is replaced by a different version between those calls.

Implementations **must** not advertise any pair of extensions that cannot be enabled together due to behavioral differences, or any extension that cannot be enabled against the advertised version.

Valid Usage (Implicit)

- VUID-vkEnumerateInstanceExtensionProperties-pLayerName-parameter
If `pLayerName` is not `NULL`, `pLayerName` **must** be a null-terminated UTF-8 string
- VUID-vkEnumerateInstanceExtensionProperties-pPropertyCount-parameter
`pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkEnumerateInstanceExtensionProperties-pProperties-parameter
If the value referenced by `pPropertyCount` is not 0, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` `VkExtensionProperties` structures

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_LAYER_NOT_PRESENT`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

32.5.2. Device Extensions

Device extensions add new [device-level functionality](#) to the API, outside of the core specification. If Vulkan 1.1 is required by the extension, it **may** also add new [physical-device-level functionality](#).

To query the extensions available to a given physical device, call:

```
// Provided by VK_VERSION_1_0
VkResult vkEnumerateDeviceExtensionProperties(
    VkPhysicalDevice          physicalDevice,
    const char*               pLayerName,
    uint32_t*                 pPropertyCount,
    VkExtensionProperties*     pProperties);
```

- `physicalDevice` is the physical device that will be queried.
- `pLayerName` is either `NULL` or a pointer to a null-terminated UTF-8 string naming the layer to retrieve extensions from.
- `pPropertyCount` is a pointer to an integer related to the number of extension properties available or queried, and is treated in the same fashion as the `vkEnumerateInstanceExtensionProperties::pPropertyCount` parameter.
- `pProperties` is either `NULL` or a pointer to an array of `VkExtensionProperties` structures.

When `pLayerName` parameter is `NULL`, only extensions provided by the Vulkan implementation or by implicitly enabled layers are returned. When `pLayerName` is the name of a layer, the device extensions provided by that layer are returned.

Implementations **must** not advertise any pair of extensions that cannot be enabled together due to behavioral differences, or any extension that cannot be enabled against the advertised version.

NOTE Due to platform details on Android, `vkEnumerateDeviceExtensionProperties` may be called with `physicalDevice` equal to `NULL` during layer discovery. This behavior will only be observed by layer implementations, and not the underlying Vulkan driver.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkEnumerateDeviceExtensionProperties` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkEnumerateDeviceExtensionProperties-physicalDevice-parameter
`physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkEnumerateDeviceExtensionProperties-pLayerName-parameter
If `pLayerName` is not `NULL`, `pLayerName` **must** be a null-terminated UTF-8 string
- VUID-vkEnumerateDeviceExtensionProperties-pPropertyCount-parameter
`pPropertyCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkEnumerateDeviceExtensionProperties-pProperties-parameter
If the value referenced by `pPropertyCount` is not 0, and `pProperties` is not `NULL`, `pProperties` **must** be a valid pointer to an array of `pPropertyCount` `VkExtensionProperties` structures

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_LAYER_NOT_PRESENT`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkExtensionProperties` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkExtensionProperties {
    char          extensionName[VK_MAX_EXTENSION_NAME_SIZE];
    uint32_t      specVersion;
} VkExtensionProperties;
```

- `extensionName` is an array of `VK_MAX_EXTENSION_NAME_SIZE` `char` containing a null-terminated UTF-8 string which is the name of the extension.
- `specVersion` is the version of this extension. It is an integer, incremented with backward compatible changes.

32.6. Extension Dependencies

Some extensions are dependent on other extensions, or on specific core API versions, to function. To enable extensions with dependencies, any *required extensions* **must** also be enabled through the same API mechanisms when creating an instance with `vkCreateInstance` or a device with `vkCreateDevice`. Each extension which has such dependencies documents them in the [appendix summarizing that extension](#).

If an extension is supported (as queried by `vkEnumerateInstanceExtensionProperties` or `vkEnumerateDeviceExtensionProperties`), then *required extensions* of that extension **must** also be supported for the same instance or physical device.

Any device extension that has an instance extension dependency that is not enabled by `vkCreateInstance` is considered to be unsupported, hence it **must** not be returned by `vkEnumerateDeviceExtensionProperties` for any `VkPhysicalDevice` child of the instance. Instance extensions do not have dependencies on device extensions.

If a required extension has been [promoted](#) to another extension or to a core API version, then as a

general rule, the dependency is also satisfied by the promoted extension or core version. This will be true so long as any features required by the original extension are also required or enabled by the promoted extension or core version. However, in some cases an extension is promoted while making some of its features optional in the promoted extension or core version. In this case, the dependency **may** not be satisfied. The only way to be certain is to look at the descriptions of the original dependency and the promoted version in the [Layers & Extensions](#) and [Core Revisions](#) appendices.

NOTE

There is metadata in `vk.xml` describing some aspects of promotion, especially `requires`, `promotedto` and `deprecatedby` attributes of `<extension>` tags. However, the metadata does not yet fully describe this scenario. In the future, we may extend the XML schema to describe the full set of extensions and versions satisfying a dependency. As discussed in more detail for [Promotion](#) below, when an extension is promoted it does not mean that a mechanical substitution of an extension API by the corresponding promoted API will work in exactly the same fashion; be supported at runtime; or even exist.

32.7. Compatibility Guarantees (Informative)

This section is marked as informal as there is no binding responsibility on implementations of the Vulkan API - these guarantees are however a contract between the Vulkan Working Group and developers using this Specification.

32.7.1. Core Versions

Each of the [major](#), [minor](#), and [patch versions](#) of the Vulkan specification provide different compatibility guarantees.

Patch Versions

A difference in the patch version indicates that a set of bug fixes or clarifications have been made to the Specification. Informative enums returned by Vulkan commands that will not affect the runtime behavior of a valid application may be added in a patch version (e.g. [VkVendorId](#)).

The specification's patch version is strictly increasing for a given major version of the specification; any change to a specification as described above will result in the patch version being increased by 1. Patch versions are applied to all minor versions, even if a given minor version is not affected by the provoking change.

Specifications with different patch versions but the same major and minor version are *fully compatible* with each other - such that a valid application written against one will work with an implementation of another.

NOTE

If a patch version includes a bug fix or clarification that could have a significant impact on developer expectations, these will be highlighted in the change log. Generally the Vulkan Working Group tries to avoid these kinds of changes, instead fixing them in either an extension or core version.

Minor Versions

Changes in the minor version of the specification indicate that new functionality has been added to the core specification. This will usually include new interfaces in the header, and **may** also include behavior changes and bug fixes. Core functionality **may** be marked as legacy in a minor version, but will not be obsoleted or removed.

The specification's minor version is strictly increasing for a given major version of the specification; any change to a specification as described above will result in the minor version being increased by 1. Changes that can be accommodated in a patch version will not increase the minor version.

Specifications with a lower minor version are *backwards compatible* with an implementation of a specification with a higher minor version for core functionality and extensions issued with the KHR vendor tag. Vendor and multi-vendor extensions are not guaranteed to remain functional across minor versions, though in general they are with few exceptions - see [Obsolescence](#) for more information.

Major Versions

A difference in the major version of specifications indicates a large set of changes which will likely include interface changes, behavioral changes, removal of [legacy functionality](#), and the modification, addition, or replacement of other functionality.

The specification's major version is monotonically increasing; any change to the specification as described above will result in the major version being increased. Changes that can be accommodated in a patch or minor version will not increase the major version.

The Vulkan Working Group intends to only issue a new major version of the Specification in order to realize significant improvements to the Vulkan API that will necessarily require breaking compatibility.

A new major version will likely include a wholly new version of the specification to be issued - which could include an overhaul of the versioning semantics for the minor and patch versions. The patch and minor versions of a specification are therefore not meaningful across major versions. If a major version of the specification includes similar versioning semantics, it is expected that the patch and the minor version will be reset to 0 for that major version.

32.7.2. Extensions

A KHR extension **must** be able to be enabled alongside any other KHR extension, and for any minor or patch version of the core Specification beyond the minimum version it requires. A multi-vendor extension **should** be able to be enabled alongside any KHR extension or other multi-vendor extension, and for any minor or patch version of the core Specification beyond the minimum version it requires. A vendor extension **should** be able to be enabled alongside any KHR extension, multi-vendor extension, or other vendor extension from the same vendor, and for any minor or patch version of the core Specification beyond the minimum version it requires. A vendor extension **may** be able to be enabled alongside vendor extensions from another vendor.

The one other exception to this is if a vendor or multi-vendor extension is [made obsolete](#) by either a

core version or another extension, which will be highlighted in the [extension appendix](#).

Promotion

Extensions, or features of an extension, **may** be promoted to a new [core version of the API](#), or a newer extension which an equal or greater number of implementors are in favor of.

NOTE

Structure and enumerated types defined in extensions that were promoted to core in a later version of Vulkan are defined in terms of the equivalent Vulkan core interfaces in that version and its successors. This affects the Vulkan Specification, the Vulkan header files, and the corresponding XML Registry.

When extension functionality is promoted, minor changes **may** be introduced, limited to the following:

- Naming
- Non-intrusive parameter changes
- [Feature advertisement/enablement](#)
- Combining structure parameters into larger structures
- Author ID suffixes changed or removed

NOTE

If extension functionality is promoted, there is no guarantee of direct compatibility, however it should require little effort to port code from the original feature to the promoted one.

The Vulkan Working Group endeavors to ensure that larger changes are marked as either [deprecated](#) or [obsoleted](#) as appropriate, and can do so retroactively if necessary.

Extensions that are promoted are listed as being promoted in their extension appendices, with reference to where they were promoted to.

When an extension is promoted, any backwards compatibility aliases which exist in the extension will **not** be promoted.

NOTE

As a hypothetical example, if the [VK_KHR_surface](#) extension were promoted to part of a future core version, the [VK_COLOR_SPACE_SRGB_NONLINEAR_KHR](#) token defined by that extension would be promoted to [VK_COLOR_SPACE_SRGB_NONLINEAR](#). However, the [VK_COLORSPACE_SRGB_NONLINEAR_KHR](#) token aliases [VK_COLOR_SPACE_SRGB_NONLINEAR_KHR](#). The [VK_COLORSPACE_SRGB_NONLINEAR_KHR](#) would not be promoted, because it is a backwards compatibility alias that exists only due to a naming mistake when the extension was initially published.

Deprecation

Extensions **may** be marked as [deprecated](#) when they are no longer intended to be used by applications under certain conditions. Generally, a new feature will become available to solve the

use case in another extension or core version of the API, but it is not guaranteed.

NOTE

Features that are intended to replace deprecated functionality have no guarantees of compatibility, and applications may require drastic modification in order to make use of the new features.

Extensions that are deprecated are listed as being deprecated in their extension appendices, with an explanation of the deprecation and any related features that are relevant.

Obsolescence

Occasionally, an extension will be marked as obsolete if a new version of the core API or a new extension is fundamentally incompatible with it. An obsoleted extension **must** not be used with the extension or core version that obsoleted it.

Extensions that are obsoleted are listed as being obsoleted in their extension appendices, with reference to what they were obsoleted by.

Aliases

When an extension is promoted or deprecated by a newer feature, some or all of its functionality **may** be replicated into the newer feature. Rather than duplication of all the documentation and definitions, the specification instead identifies the identical commands and types as *aliases* of one another. Each alias is mentioned together with the definition it aliases, with the older aliases marked as “equivalents”. Each alias of the same command has identical behavior, and each alias of the same type has identical meaning - they can be used interchangeably in an application with no compatibility issues.

NOTE

For promoted types, the aliased extension type is semantically identical to the new core type. The C99 headers simply `typedef` the older aliases to the promoted types.

For promoted command aliases, however, there are two separate command definitions, due to the fact that the C99 ABI has no way to alias command definitions without resorting to macros. Calling either command will produce identical behavior within the bounds of the specification, and should still invoke the same path in the implementation. Debug tools may use separate commands with different debug behavior; to write the appropriate command name to an output log, for instance.

Special Use Extensions

Some extensions exist only to support a specific purpose or specific class of application. These are referred to as “special use extensions”. Use of these extensions in applications not meeting the special use criteria is not recommended.

Special use cases are restricted, and only those defined below are used to describe extensions:

Table 42. Extension Special Use Cases

Special Use	XML Tag	Full Description
CAD support	cadsupport	Extension is intended to support specialized functionality used by CAD/CAM applications.
D3D support	d3demulation	Extension is intended to support D3D emulation layers, and applications ported from D3D, by adding functionality specific to D3D.
Developer tools	devtools	Extension is intended to support developer tools such as capture-replay libraries.
Debugging tools	debugging	Extension is intended for use by applications when debugging.
OpenGL / ES support	glesemulation	Extension is intended to support OpenGL and/or OpenGL ES emulation layers, and applications ported from those APIs, by adding functionality specific to those APIs.

Special use extensions are identified in the metadata for each such extension in the [Layers & Extensions](#) appendix, using the name in the “Special Use” column above.

Special use extensions are also identified in `vk.xml` with the short name in “XML Tag” column above, as described in the “API Extensions (`extension` tag)” section of the [registry schema documentation](#).

Chapter 33. Features

Features advertise additional functionality which **can** be enabled in the API. If a feature is *supported* by a physical device, it **must** be *enabled* when creating a [VkDevice](#) in order to use its associated functionality with that [VkDevice](#). If a feature is *unsupported*, that functionality **cannot** be used with that [VkDevice](#).

NOTE

Features are reported via the extensible structure [VkPhysicalDeviceFeatures2](#), which was added in the [VK_KHR_get_physical_device_properties2](#) extension and included in Vulkan 1.1. When new features are added in future Vulkan versions or extensions, each extension **should** introduce one new feature structure, if needed. This structure **can** be added to the [pNext](#) chain of the [VkPhysicalDeviceFeatures2](#) structure.

For convenience, new core versions of Vulkan **may** introduce new unified feature structures for features promoted from extensions. At the same time, the extension's original feature structure (if any) is also promoted to the core API, and is an alias of the extension's structure. This results in multiple names for the same feature: in the original extension's feature structure and the promoted structure alias, in the unified feature structure. When a feature was implicitly supported and enabled in the extension, but an explicit name was added during promotion, then the extension itself acts as an alias for the feature as listed in the table below.

All aliases of the same feature in the core API **must** be reported consistently: either all **must** be reported as supported, or none of them. When a promoted extension is available, any corresponding feature aliases **must** be supported.

Table 43. Extension Feature Aliases

Extension	Feature(s)

To query supported features, call:

WARNING

This functionality is superseded by [Vulkan Version 1.1](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
void vkGetPhysicalDeviceFeatures(
    VkPhysicalDevice          physicalDevice,
    VkPhysicalDeviceFeatures* pFeatures);
```

- [physicalDevice](#) is the physical device from which to query the supported features.
- [pFeatures](#) is a pointer to a [VkPhysicalDeviceFeatures](#) structure in which the physical device features are returned. For each feature, a value of [VK_TRUE](#) specifies that the feature is supported on this physical device, and [VK_FALSE](#) specifies that the feature is not supported.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceFeatures-physicalDevice-parameter **physicalDevice** **must** be a valid [VkPhysicalDevice](#) handle
- VUID-vkGetPhysicalDeviceFeatures-pFeatures-parameter **pFeatures** **must** be a valid pointer to a [VkPhysicalDeviceFeatures](#) structure

Fine-grained features used by a logical device **must** be enabled at [VkDevice](#) creation time. If a feature is enabled that the physical device does not support, [VkDevice](#) creation will fail and return [VK_ERROR_FEATURE_NOT_PRESENT](#).

The fine-grained features are enabled by passing a pointer to the [VkPhysicalDeviceFeatures](#) structure via the [pEnabledFeatures](#) member of the [VkDeviceCreateInfo](#) structure that is passed into the [vkCreateDevice](#) call. If a member of [pEnabledFeatures](#) is [VK_TRUE](#) or [VK_FALSE](#), then the device will be created with the indicated feature enabled or disabled, respectively. Features **can** also be enabled by using the [VkPhysicalDeviceFeatures2](#) structure.

If an application wishes to enable all features supported by a device, it **can** simply pass in the [VkPhysicalDeviceFeatures](#) structure that was previously returned by [vkGetPhysicalDeviceFeatures](#). To disable an individual feature, the application **can** set the desired member to [VK_FALSE](#) in the same structure. Setting [pEnabledFeatures](#) to [NULL](#) and not including a [VkPhysicalDeviceFeatures2](#) in the [pNext](#) chain of [VkDeviceCreateInfo](#) is equivalent to setting all members of the structure to [VK_FALSE](#).

NOTE

Some features, such as [robustBufferAccess](#), **may** incur a runtime performance cost. Application writers **should** carefully consider the implications of enabling all supported features.

To query supported features defined by the core or extensions, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceFeatures2(
    VkPhysicalDevice          physicalDevice,
    VkPhysicalDeviceFeatures2* pFeatures);
```

- **physicalDevice** is the physical device from which to query the supported features.
- **pFeatures** is a pointer to a [VkPhysicalDeviceFeatures2](#) structure in which the physical device features are returned.

Each structure in [pFeatures](#) and its [pNext](#) chain contains members corresponding to fine-grained features. Each structure in [pFeatures](#) and its [pNext](#) chain contains [VkBool32](#) members corresponding to fine-grained features. Each such member is returned with a [VK_TRUE](#) value indicating that feature is supported on this physical device, or a [VK_FALSE](#) value indicating it is unsupported.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceFeatures2-physicalDevice-parameter **physicalDevice** must be a valid [VkPhysicalDevice](#) handle
- VUID-vkGetPhysicalDeviceFeatures2-pFeatures-parameter **pFeatures** must be a valid pointer to a [VkPhysicalDeviceFeatures2](#) structure

The [VkPhysicalDeviceFeatures2](#) structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceFeatures2 {
    VkStructureType    sType;
    void*              pNext;
    VkPhysicalDeviceFeatures  features;
} VkPhysicalDeviceFeatures2;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is [NULL](#) or a pointer to a structure extending this structure.
- **features** is a [VkPhysicalDeviceFeatures](#) structure describing the fine-grained features of the Vulkan 1.0 API.

The **pNext** chain of this structure is used to extend the structure with features defined by extensions. This structure **can** be used in [vkGetPhysicalDeviceFeatures2](#) or **can** be included in the **pNext** chain of a [VkDeviceCreateInfo](#) structure, in which case it controls which features are enabled in lieu of **pEnabledFeatures**.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFeatures2-sType-sType **sType** must be [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FEATURES_2](#)

The [VkPhysicalDeviceFeatures](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPhysicalDeviceFeatures {
    VkBool32    robustBufferAccess;
    VkBool32    fullDrawIndexUint32;
    VkBool32    imageCubeArray;
    VkBool32    independentBlend;
    VkBool32    geometryShader;
    VkBool32    tessellationShader;
    VkBool32    sampleRateShading;
    VkBool32    dualSrcBlend;
    VkBool32    logicOp;
    VkBool32    multiDrawIndirect;
```

```

VkBool32    drawIndirectFirstInstance;
VkBool32    depthClamp;
VkBool32    depthBiasClamp;
VkBool32    fillModeNonSolid;
VkBool32    depthBounds;
VkBool32    wideLines;
VkBool32    largePoints;
VkBool32    alphaToOne;
VkBool32    multiViewport;
VkBool32    samplerAnisotropy;
VkBool32    textureCompressionETC2;
VkBool32    textureCompressionASTC_LDR;
VkBool32    textureCompressionBC;
VkBool32    occlusionQueryPrecise;
VkBool32    pipelineStatisticsQuery;
VkBool32    vertexPipelineStoresAndAtomics;
VkBool32    fragmentStoresAndAtomics;
VkBool32    shaderTessellationAndGeometryPointSize;
VkBool32    shaderImageGatherExtended;
VkBool32    shaderStorageImageExtendedFormats;
VkBool32    shaderStorageImageMultisample;
VkBool32    shaderStorageImageReadWithoutFormat;
VkBool32    shaderStorageImageWriteWithoutFormat;
VkBool32    shaderUniformBufferArrayDynamicIndexing;
VkBool32    shaderSampledImageArrayDynamicIndexing;
VkBool32    shaderStorageBufferArrayDynamicIndexing;
VkBool32    shaderStorageImageArrayDynamicIndexing;
VkBool32    shaderClipDistance;
VkBool32    shaderCullDistance;
VkBool32    shaderFloat64;
VkBool32    shaderInt64;
VkBool32    shaderInt16;
VkBool32    shaderResourceResidency;
VkBool32    shaderResourceMinLod;
VkBool32    sparseBinding;
VkBool32    sparseResidencyBuffer;
VkBool32    sparseResidencyImage2D;
VkBool32    sparseResidencyImage3D;
VkBool32    sparseResidency2Samples;
VkBool32    sparseResidency4Samples;
VkBool32    sparseResidency8Samples;
VkBool32    sparseResidency16Samples;
VkBool32    sparseResidencyAliased;
VkBool32    variableMultisampleRate;
VkBool32    inheritedQueries;
} VkPhysicalDeviceFeatures;

```

This structure describes the following features:

- **robustBufferAccess** enables [Robust Buffer Access](#) guarantees for shader buffer accesses.

- `fullDrawIndexUint32` specifies the full 32-bit range of indices is supported for indexed draw calls when using a `VkIndexType` of `VK_INDEX_TYPE_UINT32`. `maxDrawIndexedIndexValue` is the maximum index value that **may** be used (aside from the primitive restart index, which is always $2^{32}-1$ when the `VkIndexType` is `VK_INDEX_TYPE_UINT32`). If this feature is supported, `maxDrawIndexedIndexValue` **must** be $2^{32}-1$; otherwise it **must** be no smaller than $2^{24}-1$. See `maxDrawIndexedIndexValue`.
- `imageCubeArray` specifies whether image views with a `VkImageViewType` of `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY` **can** be created, and that the corresponding `SampledCubeArray` and `ImageCubeArray` SPIR-V capabilities **can** be used in shader code.
- `independentBlend` specifies whether the `VkPipelineColorBlendAttachmentState` settings are controlled independently per-attachment. If this feature is not enabled, the `VkPipelineColorBlendAttachmentState` settings for all color attachments **must** be identical. Otherwise, a different `VkPipelineColorBlendAttachmentState` **can** be provided for each bound color attachment.
- `geometryShader` specifies whether geometry shaders are supported. If this feature is not enabled, the `VK_SHADER_STAGE_GEOMETRY_BIT` and `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT` enum values **must** not be used. This also specifies whether shader modules **can** declare the `Geometry` capability.
- `tessellationShader` specifies whether tessellation control and evaluation shaders are supported. If this feature is not enabled, the `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT`, `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT`, `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT`, `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`, and `VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_STATE_CREATE_INFO` enum values **must** not be used. This also specifies whether shader modules **can** declare the `Tessellation` capability.
- `sampleRateShading` specifies whether `Sample Shading` and multisample interpolation are supported. If this feature is not enabled, the `sampleShadingEnable` member of the `VkPipelineMultisampleStateCreateInfo` structure **must** be `VK_FALSE` and the `minSampleShading` member is ignored. This also specifies whether shader modules **can** declare the `SampleRateShading` capability.
- `dualSrcBlend` specifies whether blend operations which take two sources are supported. If this feature is not enabled, the `VK_BLEND_FACTOR_SRC1_COLOR`, `VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR`, `VK_BLEND_FACTOR_SRC1_ALPHA`, and `VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA` enum values **must** not be used as source or destination blending factors. See [Dual-Source Blending](#).
- `logicOp` specifies whether logic operations are supported. If this feature is not enabled, the `logicOpEnable` member of the `VkPipelineColorBlendStateCreateInfo` structure **must** be `VK_FALSE`, and the `logicOp` member is ignored.
- `multiDrawIndirect` specifies whether multiple draw indirect is supported. If this feature is not enabled, the `drawCount` parameter to the `vkCmdDrawIndirect` and `vkCmdDrawIndexedIndirect` commands **must** be 0 or 1. The `maxDrawIndirectCount` member of the `VkPhysicalDeviceLimits` structure **must** also be 1 if this feature is not supported. See `maxDrawIndirectCount`.
- `drawIndirectFirstInstance` specifies whether indirect drawing calls support the `firstInstance` parameter. If this feature is not enabled, the `firstInstance` member of all `VkDrawIndirectCommand` and `VkDrawIndexedIndirectCommand` structures that are provided to the `vkCmdDrawIndirect` and `vkCmdDrawIndexedIndirect` commands **must** be 0.

- **depthClamp** specifies whether depth clamping is supported. If this feature is not enabled, the **depthClampEnable** member of the **VkPipelineRasterizationStateCreateInfo** structure **must** be **VK_FALSE**. Otherwise, setting **depthClampEnable** to **VK_TRUE** will enable depth clamping.
- **depthBiasClamp** specifies whether depth bias clamping is supported. If this feature is not enabled, the **depthBiasClamp** member of the **VkPipelineRasterizationStateCreateInfo** structure **must** be 0.0 unless the **VK_DYNAMIC_STATE_DEPTH_BIAS** dynamic state is enabled, in which case the **depthBiasClamp** parameter to **vkCmdSetDepthBias** **must** be 0.0.
- **fillModeNonSolid** specifies whether point and wireframe fill modes are supported. If this feature is not enabled, the **VK_POLYGON_MODE_POINT** and **VK_POLYGON_MODE_LINE** enum values **must** not be used.
- **depthBounds** specifies whether depth bounds tests are supported. If this feature is not enabled, the **depthBoundsTestEnable** member of the **VkPipelineDepthStencilStateCreateInfo** structure **must** be **VK_FALSE** unless the **VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE** dynamic state is enabled, in which case the **depthBoundsTestEnable** parameter to **vkCmdSetDepthBoundsTestEnableEXT** **must** be **VK_FALSE**. When **depthBoundsTestEnable** is **VK_FALSE**, the **minDepthBounds** and **maxDepthBounds** members of the **VkPipelineDepthStencilStateCreateInfo** structure are ignored.
- **wideLines** specifies whether lines with width other than 1.0 are supported. If this feature is not enabled, the **lineWidth** member of the **VkPipelineRasterizationStateCreateInfo** structure **must** be 1.0 unless the **VK_DYNAMIC_STATE_LINE_WIDTH** dynamic state is enabled, in which case the **lineWidth** parameter to **vkCmdSetLineWidth** **must** be 1.0. When this feature is supported, the range and granularity of supported line widths are indicated by the **lineWidthRange** and **lineWidthGranularity** members of the **VkPhysicalDeviceLimits** structure, respectively.
- **largePoints** specifies whether points with size greater than 1.0 are supported. If this feature is not enabled, only a point size of 1.0 written by a shader is supported. The range and granularity of supported point sizes are indicated by the **pointSizeRange** and **pointSizeGranularity** members of the **VkPhysicalDeviceLimits** structure, respectively.
- **alphaToOne** specifies whether the implementation is able to replace the alpha value of the fragment shader color output in the **Multisample Coverage** fragment operation. If this feature is not enabled, then the **alphaToOneEnable** member of the **VkPipelineMultisampleStateCreateInfo** structure **must** be **VK_FALSE**. Otherwise setting **alphaToOneEnable** to **VK_TRUE** will enable alpha-to-one behavior.
- **multiViewport** specifies whether more than one viewport is supported. If this feature is not enabled:
 - The **viewportCount** and **scissorCount** members of the **VkPipelineViewportStateCreateInfo** structure **must** be 1.
 - The **firstViewport** and **viewportCount** parameters to the **vkCmdSetViewport** command **must** be 0 and 1, respectively.
 - The **firstScissor** and **scissorCount** parameters to the **vkCmdSetScissor** command **must** be 0 and 1, respectively.
- **samplerAnisotropy** specifies whether anisotropic filtering is supported. If this feature is not enabled, the **anisotropyEnable** member of the **VkSamplerCreateInfo** structure **must** be **VK_FALSE**.
- **textureCompressionETC2** specifies whether all of the ETC2 and EAC compressed texture formats

are supported. If this feature is enabled, then the `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`, `VK_FORMAT_FEATURE_BLIT_SRC_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` features **must** be supported in `optimalTilingFeatures` for the following formats:

- `VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK`
- `VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK`
- `VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK`
- `VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK`
- `VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK`
- `VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK`
- `VK_FORMAT_EAC_R11_UNORM_BLOCK`
- `VK_FORMAT_EAC_R11_SNORM_BLOCK`
- `VK_FORMAT_EAC_R11G11_UNORM_BLOCK`
- `VK_FORMAT_EAC_R11G11_SNORM_BLOCK`

To query for additional properties, or if the feature is not enabled, `vkGetPhysicalDeviceFormatProperties` and `vkGetPhysicalDeviceImageFormatProperties` can be used to check for supported properties of individual formats as normal.

- `textureCompressionASTC_LDR` specifies whether all of the ASTC LDR compressed texture formats are supported. If this feature is enabled, then the `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`, `VK_FORMAT_FEATURE_BLIT_SRC_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` features **must** be supported in `optimalTilingFeatures` for the following formats:

- `VK_FORMAT_ASTC_4x4_UNORM_BLOCK`
- `VK_FORMAT_ASTC_4x4_SRGB_BLOCK`
- `VK_FORMAT_ASTC_5x4_UNORM_BLOCK`
- `VK_FORMAT_ASTC_5x4_SRGB_BLOCK`
- `VK_FORMAT_ASTC_5x5_UNORM_BLOCK`
- `VK_FORMAT_ASTC_5x5_SRGB_BLOCK`
- `VK_FORMAT_ASTC_6x5_UNORM_BLOCK`
- `VK_FORMAT_ASTC_6x5_SRGB_BLOCK`
- `VK_FORMAT_ASTC_6x6_UNORM_BLOCK`
- `VK_FORMAT_ASTC_6x6_SRGB_BLOCK`
- `VK_FORMAT_ASTC_8x5_UNORM_BLOCK`
- `VK_FORMAT_ASTC_8x5_SRGB_BLOCK`
- `VK_FORMAT_ASTC_8x6_UNORM_BLOCK`
- `VK_FORMAT_ASTC_8x6_SRGB_BLOCK`
- `VK_FORMAT_ASTC_8x8_UNORM_BLOCK`
- `VK_FORMAT_ASTC_8x8_SRGB_BLOCK`

- VK_FORMAT_ASTC_10x5_UNORM_BLOCK
- VK_FORMAT_ASTC_10x5_SRGB_BLOCK
- VK_FORMAT_ASTC_10x6_UNORM_BLOCK
- VK_FORMAT_ASTC_10x6_SRGB_BLOCK
- VK_FORMAT_ASTC_10x8_UNORM_BLOCK
- VK_FORMAT_ASTC_10x8_SRGB_BLOCK
- VK_FORMAT_ASTC_10x10_UNORM_BLOCK
- VK_FORMAT_ASTC_10x10_SRGB_BLOCK
- VK_FORMAT_ASTC_12x10_UNORM_BLOCK
- VK_FORMAT_ASTC_12x10_SRGB_BLOCK
- VK_FORMAT_ASTC_12x12_UNORM_BLOCK
- VK_FORMAT_ASTC_12x12_SRGB_BLOCK

To query for additional properties, or if the feature is not enabled, [vkGetPhysicalDeviceFormatProperties](#) and [vkGetPhysicalDeviceImageFormatProperties](#) **can** be used to check for supported properties of individual formats as normal.

- **textureCompressionBC** specifies whether all of the BC compressed texture formats are supported. If this feature is enabled, then the **VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT**, **VK_FORMAT_FEATURE_BLIT_SRC_BIT** and **VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT** features **must** be supported in **optimalTilingFeatures** for the following formats:

- VK_FORMAT_BC1_RGB_UNORM_BLOCK
- VK_FORMAT_BC1_RGB_SRGB_BLOCK
- VK_FORMAT_BC1_RGBA_UNORM_BLOCK
- VK_FORMAT_BC1_RGBA_SRGB_BLOCK
- VK_FORMAT_BC2_UNORM_BLOCK
- VK_FORMAT_BC2_SRGB_BLOCK
- VK_FORMAT_BC3_UNORM_BLOCK
- VK_FORMAT_BC3_SRGB_BLOCK
- VK_FORMAT_BC4_UNORM_BLOCK
- VK_FORMAT_BC4_SNORM_BLOCK
- VK_FORMAT_BC5_UNORM_BLOCK
- VK_FORMAT_BC5_SNORM_BLOCK
- VK_FORMAT_BC6H_UFLOAT_BLOCK
- VK_FORMAT_BC6H_SFLOAT_BLOCK
- VK_FORMAT_BC7_UNORM_BLOCK
- VK_FORMAT_BC7_SRGB_BLOCK

To query for additional properties, or if the feature is not enabled, [vkGetPhysicalDeviceFormatProperties](#) and [vkGetPhysicalDeviceImageFormatProperties](#) **can** be used to check for supported properties of individual formats as normal.

- [occlusionQueryPrecise](#) specifies whether occlusion queries returning actual sample counts are supported. Occlusion queries are created in a [VkQueryPool](#) by specifying the [queryType](#) of [VK_QUERY_TYPE_OCCLUSION](#) in the [VkQueryPoolCreateInfo](#) structure which is passed to [vkCreateQueryPool](#). If this feature is enabled, queries of this type **can** enable [VK_QUERY_CONTROL_PRECISE_BIT](#) in the [flags](#) parameter to [vkCmdBeginQuery](#). If this feature is not supported, the implementation supports only boolean occlusion queries. When any samples are passed, boolean queries will return a non-zero result value, otherwise a result value of zero is returned. When this feature is enabled and [VK_QUERY_CONTROL_PRECISE_BIT](#) is set, occlusion queries will report the actual number of samples passed.
- [pipelineStatisticsQuery](#) specifies whether the pipeline statistics queries are supported. If this feature is not enabled, queries of type [VK_QUERY_TYPE_PIPELINE_STATISTICS](#) **cannot** be created, and none of the [VkQueryPipelineStatisticFlagBits](#) bits **can** be set in the [pipelineStatistics](#) member of the [VkQueryPoolCreateInfo](#) structure.
- [vertexPipelineStoresAndAtomics](#) specifies whether storage buffers and images support stores and atomic operations in the vertex, tessellation, and geometry shader stages. If this feature is not enabled, all storage image, storage texel buffer, and storage buffer variables used by these stages in shader modules **must** be decorated with the [NonWritable](#) decoration (or the [readonly](#) memory qualifier in GLSL).
- [fragmentStoresAndAtomics](#) specifies whether storage buffers and images support stores and atomic operations in the fragment shader stage. If this feature is not enabled, all storage image, storage texel buffer, and storage buffer variables used by the fragment stage in shader modules **must** be decorated with the [NonWritable](#) decoration (or the [readonly](#) memory qualifier in GLSL).
- [shaderTessellationAndGeometryPointSize](#) specifies whether the [PointSize](#) built-in decoration is available in the tessellation control, tessellation evaluation, and geometry shader stages. If this feature is not enabled, members decorated with the [PointSize](#) built-in decoration **must** not be read from or written to and all points written from a tessellation or geometry shader will have a size of 1.0. This also specifies whether shader modules **can** declare the [TessellationPointSize](#) capability for tessellation control and evaluation shaders, or if the shader modules **can** declare the [GeometryPointSize](#) capability for geometry shaders. An implementation supporting this feature **must** also support one or both of the [tessellationShader](#) or [geometryShader](#) features.
- [shaderImageGatherExtended](#) specifies whether the extended set of image gather instructions are available in shader code. If this feature is not enabled, the [OpImage*Gather](#) instructions do not support the [Offset](#) and [ConstOffsets](#) operands. This also specifies whether shader modules **can** declare the [ImageGatherExtended](#) capability.
- [shaderStorageImageExtendedFormats](#) specifies whether all the “storage image extended formats” below are supported; if this feature is supported, then the [VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT](#) **must** be supported in [optimalTilingFeatures](#) for the following formats:
 - [VK_FORMAT_R16G16_SFLOAT](#)
 - [VK_FORMAT_B10G11R11_UFLOAT_PACK32](#)
 - [VK_FORMAT_R16_SFLOAT](#)

- `VK_FORMAT_R16G16B16A16_UNORM`
- `VK_FORMAT_A2B10G10R10_UNORM_PACK32`
- `VK_FORMAT_R16G16_UNORM`
- `VK_FORMAT_R8G8_UNORM`
- `VK_FORMAT_R16_UNORM`
- `VK_FORMAT_R8_UNORM`
- `VK_FORMAT_R16G16B16A16_SNORM`
- `VK_FORMAT_R16G16_SNORM`
- `VK_FORMAT_R8G8_SNORM`
- `VK_FORMAT_R16_SNORM`
- `VK_FORMAT_R8_SNORM`
- `VK_FORMAT_R16G16_SINT`
- `VK_FORMAT_R8G8_SINT`
- `VK_FORMAT_R16_SINT`
- `VK_FORMAT_R8_SINT`
- `VK_FORMAT_A2B10G10R10_UINT_PACK32`
- `VK_FORMAT_R16G16_UINT`
- `VK_FORMAT_R8G8_UINT`
- `VK_FORMAT_R16_UINT`
- `VK_FORMAT_R8_UINT`

NOTE

`shaderStorageImageExtendedFormats` feature only adds a guarantee of format support, which is specified for the whole physical device. Therefore enabling or disabling the feature via `vkCreateDevice` has no practical effect.

To query for additional properties, or if the feature is not supported, `vkGetPhysicalDeviceFormatProperties` and `vkGetPhysicalDeviceImageFormatProperties` **can** be used to check for supported properties of individual formats, as usual rules allow.

`VK_FORMAT_R32G32_UINT`, `VK_FORMAT_R32G32_SINT`, and `VK_FORMAT_R32G32_SFLOAT` from `StorageImageExtendedFormats` SPIR-V capability, are already covered by core Vulkan **mandatory format support**.

- `shaderStorageImageMultisample` specifies whether multisampled storage images are supported. If this feature is not enabled, images that are created with a `usage` that includes `VK_IMAGE_USAGE_STORAGE_BIT` **must** be created with `samples` equal to `VK_SAMPLE_COUNT_1_BIT`. This also specifies whether shader modules **can** declare the `StorageImageMultisample` and `ImageMSArray` capabilities.
- `shaderStorageImageReadWithoutFormat` specifies whether storage images and storage texel buffers

require a format qualifier to be specified when reading.

- **shaderStorageImageWriteWithoutFormat** specifies whether storage images and storage texel buffers require a format qualifier to be specified when writing.
- **shaderUniformBufferArrayDynamicIndexing** specifies whether arrays of uniform buffers **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of **VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER** or **VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC** **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also specifies whether shader modules **can** declare the **UniformBufferArrayDynamicIndexing** capability.
- **shaderSampledImageArrayDynamicIndexing** specifies whether arrays of samplers or sampled images **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of **VK_DESCRIPTOR_TYPE_SAMPLER**, **VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER**, or **VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE** **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also specifies whether shader modules **can** declare the **SampledImageArrayDynamicIndexing** capability.
- **shaderStorageBufferArrayDynamicIndexing** specifies whether arrays of storage buffers **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of **VK_DESCRIPTOR_TYPE_STORAGE_BUFFER** or **VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC** **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also specifies whether shader modules **can** declare the **StorageBufferArrayDynamicIndexing** capability.
- **shaderStorageImageArrayDynamicIndexing** specifies whether arrays of storage images **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of **VK_DESCRIPTOR_TYPE_STORAGE_IMAGE** **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also specifies whether shader modules **can** declare the **StorageImageArrayDynamicIndexing** capability.
- **shaderClipDistance** specifies whether clip distances are supported in shader code. If this feature is not enabled, any members decorated with the **ClipDistance** built-in decoration **must** not be read from or written to in shader modules. This also specifies whether shader modules **can** declare the **ClipDistance** capability.
- **shaderCullDistance** specifies whether cull distances are supported in shader code. If this feature is not enabled, any members decorated with the **CullDistance** built-in decoration **must** not be read from or written to in shader modules. This also specifies whether shader modules **can** declare the **CullDistance** capability.
- **shaderFloat64** specifies whether 64-bit floats (doubles) are supported in shader code. If this feature is not enabled, 64-bit floating-point types **must** not be used in shader code. This also specifies whether shader modules **can** declare the **Float64** capability. Declaring and using 64-bit floats is enabled for all storage classes that SPIR-V allows with the **Float64** capability.
- **shaderInt64** specifies whether 64-bit integers (signed and unsigned) are supported in shader code. If this feature is not enabled, 64-bit integer types **must** not be used in shader code. This

also specifies whether shader modules **can** declare the **Int64** capability. Declaring and using 64-bit integers is enabled for all storage classes that SPIR-V allows with the **Int64** capability.

- **shaderInt16** specifies whether 16-bit integers (signed and unsigned) are supported in shader code. If this feature is not enabled, 16-bit integer types **must** not be used in shader code. This also specifies whether shader modules **can** declare the **Int16** capability. However, this only enables a subset of the storage classes that SPIR-V allows for the **Int16** SPIR-V capability: Declaring and using 16-bit integers in the **Private**, **Workgroup**, and **Function** storage classes is enabled, while declaring them in the interface storage classes (e.g., **UniformConstant**, **Uniform**, **StorageBuffer**, **Input**, **Output**, and **PushConstant**) is not enabled.
- **shaderResourceResidency** specifies whether image operations that return resource residency information are supported in shader code. If this feature is not enabled, the **OpImageSparse*** instructions **must** not be used in shader code. This also specifies whether shader modules **can** declare the **SparseResidency** capability. The feature requires at least one of the **sparseResidency*** features to be supported. This **must** be **VK_FALSE** in Vulkan SC [SCID-8].
- **shaderResourceMinLod** specifies whether image operations specifying the minimum resource LOD are supported in shader code. If this feature is not enabled, the **MinLod** image operand **must** not be used in shader code. This also specifies whether shader modules **can** declare the **MinLod** capability.
- **sparseBinding** specifies whether resource memory **can** be managed at opaque sparse block level instead of at the object level. If this feature is not enabled, resource memory **must** be bound only on a per-object basis using the **vkBindBufferMemory** and **vkBindImageMemory** commands. In this case, buffers and images **must** not be created with **VK_BUFFER_CREATE_SPARSE_BINDING_BIT** and **VK_IMAGE_CREATE_SPARSE_BINDING_BIT** set in the **flags** member of the **VkBufferCreateInfo** and **VkImageCreateInfo** structures, respectively. Otherwise resource memory **can** be managed as described in **Sparse Resource Features**. This **must** be **VK_FALSE** in Vulkan SC [SCID-8].
- **sparseResidencyBuffer** specifies whether the device **can** access partially resident buffers. If this feature is not enabled, buffers **must** not be created with **VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT** set in the **flags** member of the **VkBufferCreateInfo** structure. This **must** be **VK_FALSE** in Vulkan SC [SCID-8].
- **sparseResidencyImage2D** specifies whether the device **can** access partially resident 2D images with 1 sample per pixel. If this feature is not enabled, images with an **imageType** of **VK_IMAGE_TYPE_2D** and **samples** set to **VK_SAMPLE_COUNT_1_BIT** **must** not be created with **VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT** set in the **flags** member of the **VkImageCreateInfo** structure. This **must** be **VK_FALSE** in Vulkan SC [SCID-8].
- **sparseResidencyImage3D** specifies whether the device **can** access partially resident 3D images. If this feature is not enabled, images with an **imageType** of **VK_IMAGE_TYPE_3D** **must** not be created with **VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT** set in the **flags** member of the **VkImageCreateInfo** structure. This **must** be **VK_FALSE** in Vulkan SC [SCID-8].
- **sparseResidency2Samples** specifies whether the physical device **can** access partially resident 2D images with 2 samples per pixel. If this feature is not enabled, images with an **imageType** of **VK_IMAGE_TYPE_2D** and **samples** set to **VK_SAMPLE_COUNT_2_BIT** **must** not be created with **VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT** set in the **flags** member of the **VkImageCreateInfo** structure. This **must** be **VK_FALSE** in Vulkan SC [SCID-8].

- `sparseResidency4Samples` specifies whether the physical device **can** access partially resident 2D images with 4 samples per pixel. If this feature is not enabled, images with an `imageType` of `VK_IMAGE_TYPE_2D` and `samples` set to `VK_SAMPLE_COUNT_4_BIT` **must** not be created with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` set in the `flags` member of the `VkImageCreateInfo` structure. This **must** be `VK_FALSE` in Vulkan SC [SCID-8].
- `sparseResidency8Samples` specifies whether the physical device **can** access partially resident 2D images with 8 samples per pixel. If this feature is not enabled, images with an `imageType` of `VK_IMAGE_TYPE_2D` and `samples` set to `VK_SAMPLE_COUNT_8_BIT` **must** not be created with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` set in the `flags` member of the `VkImageCreateInfo` structure. This **must** be `VK_FALSE` in Vulkan SC [SCID-8].
- `sparseResidency16Samples` specifies whether the physical device **can** access partially resident 2D images with 16 samples per pixel. If this feature is not enabled, images with an `imageType` of `VK_IMAGE_TYPE_2D` and `samples` set to `VK_SAMPLE_COUNT_16_BIT` **must** not be created with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` set in the `flags` member of the `VkImageCreateInfo` structure. This **must** be `VK_FALSE` in Vulkan SC [SCID-8].
- `sparseResidencyAliased` specifies whether the physical device **can** correctly access data aliased into multiple locations. If this feature is not enabled, the `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT` and `VK_IMAGE_CREATE_SPARSE_ALIASED_BIT` enum values **must** not be used in `flags` members of the `VkBufferCreateInfo` and `VkImageCreateInfo` structures, respectively. This **must** be `VK_FALSE` in Vulkan SC [SCID-8].
- `variableMultisampleRate` specifies whether all pipelines that will be bound to a command buffer during a subpass which uses no attachments **must** have the same value for `VkPipelineMultisampleStateCreateInfo::rasterizationSamples`. If set to `VK_TRUE`, the implementation supports variable multisample rates in a subpass which uses no attachments. If set to `VK_FALSE`, then all pipelines bound in such a subpass **must** have the same multisample rate. This has no effect in situations where a subpass uses any attachments.
- `inheritedQueries` specifies whether a secondary command buffer **may** be executed while a query is active.

The `VkPhysicalDeviceVulkan11Features` structure is defined as:


```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceVulkan11Features {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           storageBuffer16BitAccess;
    VkBool32           uniformAndStorageBuffer16BitAccess;
    VkBool32           storagePushConstant16;
    VkBool32           storageInputOutput16;
    VkBool32           multiview;
    VkBool32           multiviewGeometryShader;
    VkBool32           multiviewTessellationShader;
    VkBool32           variablePointersStorageBuffer;
    VkBool32           variablePointers;
    VkBool32           protectedMemory;
    VkBool32           samplerYcbcrConversion;
    VkBool32           shaderDrawParameters;
} VkPhysicalDeviceVulkan11Features;
```

This structure describes the following features:

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **storageBuffer16BitAccess** specifies whether objects in the **StorageBuffer**, or **PhysicalStorageBuffer** storage class with the **Block** decoration **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or 16-bit floating-point members **must** not be used in such objects. This also specifies whether shader modules **can** declare the **StorageBuffer16BitAccess** capability.
- **uniformAndStorageBuffer16BitAccess** specifies whether objects in the **Uniform** storage class with the **Block** decoration **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or 16-bit floating-point members **must** not be used in such objects. This also specifies whether shader modules **can** declare the **UniformAndStorageBuffer16BitAccess** capability.
- **storagePushConstant16** specifies whether objects in the **PushConstant** storage class **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or floating-point members **must** not be used in such objects. This also specifies whether shader modules **can** declare the **StoragePushConstant16** capability.
- **storageInputOutput16** specifies whether objects in the **Input** and **Output** storage classes **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or 16-bit floating-point members **must** not be used in such objects. This also specifies whether shader modules **can** declare the **StorageInputOutput16** capability.
- **multiview** specifies whether the implementation supports multiview rendering within a render pass. If this feature is not enabled, the view mask of each subpass **must** always be zero.
- **multiviewGeometryShader** specifies whether the implementation supports multiview rendering within a render pass, with **geometry shaders**. If this feature is not enabled, then a pipeline compiled against a subpass with a non-zero view mask **must** not include a geometry shader.

- `multiviewTessellationShader` specifies whether the implementation supports multiview rendering within a render pass, with `tessellation shaders`. If this feature is not enabled, then a pipeline compiled against a subpass with a non-zero view mask **must** not include any tessellation shaders.
- `variablePointersStorageBuffer` specifies whether the implementation supports the SPIR-V `VariablePointersStorageBuffer` capability. When this feature is not enabled, shader modules **must** not declare the `SPV_KHR_variable_pointers` extension or the `VariablePointersStorageBuffer` capability.
- `variablePointers` specifies whether the implementation supports the SPIR-V `VariablePointers` capability. When this feature is not enabled, shader modules **must** not declare the `VariablePointers` capability.
- `protectedMemory` specifies whether `protected memory` is supported.
- `samplerYcbcrConversion` specifies whether the implementation supports `sampler Y'CBCR conversion`. If `samplerYcbcrConversion` is `VK_FALSE`, sampler Y'C_BC_R conversion is not supported, and samplers using sampler Y'C_BC_R conversion **must** not be used.
- `shaderDrawParameters` specifies whether the implementation supports the SPIR-V `DrawParameters` capability. When this feature is not enabled, shader modules **must** not declare the `SPV_KHR_shader_draw_parameters` extension or the `DrawParameters` capability.

If the `VkPhysicalDeviceVulkan11Features` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceVulkan11Features`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- `VOID-VkPhysicalDeviceVulkan11Features-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_1_FEATURES`

The `VkPhysicalDeviceVulkan12Features` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceVulkan12Features {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           samplerMirrorClampToEdge;
    VkBool32           drawIndirectCount;
    VkBool32           storageBuffer8BitAccess;
    VkBool32           uniformAndStorageBuffer8BitAccess;
    VkBool32           storagePushConstant8;
    VkBool32           shaderBufferInt64Atomics;
    VkBool32           shaderSharedInt64Atomics;
    VkBool32           shaderFloat16;
```

```

VkBool32    shaderInt8;
VkBool32    descriptorIndexing;
VkBool32    shaderInputAttachmentArrayDynamicIndexing;
VkBool32    shaderUniformTexelBufferArrayDynamicIndexing;
VkBool32    shaderStorageTexelBufferArrayDynamicIndexing;
VkBool32    shaderUniformBufferArrayNonUniformIndexing;
VkBool32    shaderSampledImageArrayNonUniformIndexing;
VkBool32    shaderStorageBufferArrayNonUniformIndexing;
VkBool32    shaderStorageImageArrayNonUniformIndexing;
VkBool32    shaderInputAttachmentArrayNonUniformIndexing;
VkBool32    shaderUniformTexelBufferArrayNonUniformIndexing;
VkBool32    shaderStorageTexelBufferArrayNonUniformIndexing;
VkBool32    descriptorBindingUniformBufferUpdateAfterBind;
VkBool32    descriptorBindingSampledImageUpdateAfterBind;
VkBool32    descriptorBindingStorageImageUpdateAfterBind;
VkBool32    descriptorBindingStorageBufferUpdateAfterBind;
VkBool32    descriptorBindingUniformTexelBufferUpdateAfterBind;
VkBool32    descriptorBindingStorageTexelBufferUpdateAfterBind;
VkBool32    descriptorBindingUpdateUnusedWhilePending;
VkBool32    descriptorBindingPartiallyBound;
VkBool32    descriptorBindingVariableDescriptorCount;
VkBool32    runtimeDescriptorArray;
VkBool32    samplerFilterMinmax;
VkBool32    scalarBlockLayout;
VkBool32    imagelessFramebuffer;
VkBool32    uniformBufferStandardLayout;
VkBool32    shaderSubgroupExtendedTypes;
VkBool32    separateDepthStencilLayouts;
VkBool32    hostQueryReset;
VkBool32    timelineSemaphore;
VkBool32    bufferDeviceAddress;
VkBool32    bufferDeviceAddressCaptureReplay;
VkBool32    bufferDeviceAddressMultiDevice;
VkBool32    vulkanMemoryModel;
VkBool32    vulkanMemoryModelDeviceScope;
VkBool32    vulkanMemoryModelAvailabilityVisibilityChains;
VkBool32    shaderOutputViewportIndex;
VkBool32    shaderOutputLayer;
VkBool32    subgroupBroadcastDynamicId;
} VkPhysicalDeviceVulkan12Features;

```

This structure describes the following features:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **samplerMirrorClampToEdge** indicates whether the implementation supports the **VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE** sampler address mode. If this feature is not enabled, the **VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE** sampler address mode **must** not be used.

- `drawIndirectCount` indicates whether the implementation supports the `vkCmdDrawIndirectCount` and `vkCmdDrawIndexedIndirectCount` functions. If this feature is not enabled, these functions **must** not be used.
- `storageBuffer8BitAccess` indicates whether objects in the `StorageBuffer`, or `PhysicalStorageBuffer` storage class with the `Block` decoration **can** have 8-bit integer members. If this feature is not enabled, 8-bit integer members **must** not be used in such objects. This also indicates whether shader modules **can** declare the `StorageBuffer8BitAccess` capability.
- `uniformAndStorageBuffer8BitAccess` indicates whether objects in the `Uniform` storage class with the `Block` decoration **can** have 8-bit integer members. If this feature is not enabled, 8-bit integer members **must** not be used in such objects. This also indicates whether shader modules **can** declare the `UniformAndStorageBuffer8BitAccess` capability.
- `storagePushConstant8` indicates whether objects in the `PushConstant` storage class **can** have 8-bit integer members. If this feature is not enabled, 8-bit integer members **must** not be used in such objects. This also indicates whether shader modules **can** declare the `StoragePushConstant8` capability.
- `shaderBufferInt64Atomics` indicates whether shaders **can** perform 64-bit unsigned and signed integer atomic operations on buffers.
- `shaderSharedInt64Atomics` indicates whether shaders **can** perform 64-bit unsigned and signed integer atomic operations on shared memory.
- `shaderFloat16` indicates whether 16-bit floats (halves) are supported in shader code. This also indicates whether shader modules **can** declare the `Float16` capability. However, this only enables a subset of the storage classes that SPIR-V allows for the `Float16` SPIR-V capability: Declaring and using 16-bit floats in the `Private`, `Workgroup`, and `Function` storage classes is enabled, while declaring them in the interface storage classes (e.g., `UniformConstant`, `Uniform`, `StorageBuffer`, `Input`, `Output`, and `PushConstant`) is not enabled.
- `shaderInt8` indicates whether 8-bit integers (signed and unsigned) are supported in shader code. This also indicates whether shader modules **can** declare the `Int8` capability. However, this only enables a subset of the storage classes that SPIR-V allows for the `Int8` SPIR-V capability: Declaring and using 8-bit integers in the `Private`, `Workgroup`, and `Function` storage classes is enabled, while declaring them in the interface storage classes (e.g., `UniformConstant`, `Uniform`, `StorageBuffer`, `Input`, `Output`, and `PushConstant`) is not enabled.
- `descriptorIndexing` indicates whether the implementation supports the minimum set of descriptor indexing features as described in the [Feature Requirements](#) section. Enabling this feature when `vkCreateDevice` is called does not imply the other minimum descriptor indexing features are also enabled. Those other descriptor indexing features **must** be enabled individually as needed by the application.
- `shaderInputAttachmentArrayDynamicIndexing` indicates whether arrays of input attachments **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `InputAttachmentArrayDynamicIndexing` capability.
- `shaderUniformTexelBufferArrayDynamicIndexing` indicates whether arrays of uniform texel buffers **can** be indexed by integer expressions that are dynamically uniform within either the

subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `UniformTexelBufferArrayDynamicIndexing` capability.

- `shaderStorageTexelBufferArrayDynamicIndexing` indicates whether arrays of storage texel buffers **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `StorageTexelBufferArrayDynamicIndexing` capability.
- `shaderUniformBufferArrayNonUniformIndexing` indicates whether arrays of uniform buffers **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `UniformBufferArrayNonUniformIndexing` capability.
- `shaderSampledImageArrayNonUniformIndexing` indicates whether arrays of samplers or sampled images **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_SAMPLER`, `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, or `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `SampledImageArrayNonUniformIndexing` capability.
- `shaderStorageBufferArrayNonUniformIndexing` indicates whether arrays of storage buffers **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `StorageBufferArrayNonUniformIndexing` capability.
- `shaderStorageImageArrayNonUniformIndexing` indicates whether arrays of storage images **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `StorageImageArrayNonUniformIndexing` capability.
- `shaderInputAttachmentArrayNonUniformIndexing` indicates whether arrays of input attachments **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `InputAttachmentArrayNonUniformIndexing` capability.
- `shaderUniformTexelBufferArrayNonUniformIndexing` indicates whether arrays of uniform texel buffers **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code.

This also indicates whether shader modules **can** declare the `UniformTexelBufferArrayNonUniformIndexing` capability.

- `shaderStorageTexelBufferArrayNonUniformIndexing` indicates whether arrays of storage texel buffers **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `StorageTexelBufferArrayNonUniformIndexing` capability.
- `descriptorBindingUniformBufferUpdateAfterBind` indicates whether the implementation supports updating uniform buffer descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER`.
- `descriptorBindingSampledImageUpdateAfterBind` indicates whether the implementation supports updating sampled image descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_SAMPLER`, `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, or `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`.
- `descriptorBindingStorageImageUpdateAfterBind` indicates whether the implementation supports updating storage image descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`.
- `descriptorBindingStorageBufferUpdateAfterBind` indicates whether the implementation supports updating storage buffer descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER`.
- `descriptorBindingUniformTexelBufferUpdateAfterBind` indicates whether the implementation supports updating uniform texel buffer descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER`.
- `descriptorBindingStorageTexelBufferUpdateAfterBind` indicates whether the implementation supports updating storage texel buffer descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`.
- `descriptorBindingUpdateUnusedWhilePending` indicates whether the implementation supports updating descriptors while the set is in use. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT` **must** not be used.
- `descriptorBindingPartiallyBound` indicates whether the implementation supports statically using a descriptor set binding in which some descriptors are not valid. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT` **must** not be used.
- `descriptorBindingVariableDescriptorCount` indicates whether the implementation supports descriptor sets with a variable-sized last binding. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT` **must** not be used.
- `runtimeDescriptorArray` indicates whether the implementation supports the SPIR-V

`RuntimeDescriptorArray` capability. If this feature is not enabled, descriptors **must** not be declared in runtime arrays.

- `samplerFilterMinmax` indicates whether the implementation supports a minimum set of required formats supporting min/max filtering as defined by the `filterMinmaxSingleComponentFormats` property minimum requirements. If this feature is not enabled, then `VkSamplerReductionModeCreateInfo` **must** only use `VK_SAMPLER_REDUCTION_MODE_WEIGHTED_AVERAGE`.
- `scalarBlockLayout` indicates that the implementation supports the layout of resource blocks in shaders using `scalar alignment`.
- `imagelessFramebuffer` indicates that the implementation supports specifying the image view for attachments at render pass begin time via `VkRenderPassAttachmentBeginInfo`.
- `uniformBufferStandardLayout` indicates that the implementation supports the same layouts for uniform buffers as for storage and other kinds of buffers. See `Standard Buffer Layout`.
- `shaderSubgroupExtendedTypes` is a boolean specifying whether subgroup operations can use 8-bit integer, 16-bit integer, 64-bit integer, 16-bit floating-point, and vectors of these types in `group operations` with `subgroup scope`, if the implementation supports the types.
- `separateDepthStencilLayouts` indicates whether the implementation supports a `VkImageMemoryBarrier` for a depth/stencil image with only one of `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT` set, and whether `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`, or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL` can be used.
- `hostQueryReset` indicates that the implementation supports resetting queries from the host with `vkResetQueryPool`.
- `timelineSemaphore` indicates whether semaphores created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE` are supported.
- `bufferDeviceAddress` indicates that the implementation supports accessing buffer memory in shaders as storage buffers via an address queried from `vkGetBufferDeviceAddress`.
- `bufferDeviceAddressCaptureReplay` indicates that the implementation supports saving and reusing buffer and device addresses, e.g. for trace capture and replay.
- `bufferDeviceAddressMultiDevice` indicates that the implementation supports the `bufferDeviceAddress` feature for logical devices created with multiple physical devices. If this feature is not supported, buffer addresses **must** not be queried on a logical device created with more than one physical device.
- `vulkanMemoryModel` indicates whether shader modules **can** declare the `VulkanMemoryModel` capability.
- `vulkanMemoryModelDeviceScope` indicates whether the Vulkan Memory Model can use `Device` scope synchronization. This also indicates whether shader modules **can** declare the `VulkanMemoryModelDeviceScope` capability.
- `vulkanMemoryModelAvailabilityVisibilityChains` indicates whether the Vulkan Memory Model can use `availability and visibility chains` with more than one element.
- `shaderOutputViewportIndex` indicates whether the implementation supports the `ShaderViewportIndex` SPIR-V capability enabling variables decorated with the `ViewportIndex` built-

in to be exported from vertex or tessellation evaluation shaders. If this feature is not enabled, the `ViewportIndex` built-in decoration **must** not be used on outputs in vertex or tessellation evaluation shaders.

- `shaderOutputLayer` indicates whether the implementation supports the `ShaderLayer` SPIR-V capability enabling variables decorated with the `Layer` built-in to be exported from vertex or tessellation evaluation shaders. If this feature is not enabled, the `Layer` built-in decoration **must** not be used on outputs in vertex or tessellation evaluation shaders.
- If `subgroupBroadcastDynamicId` is `VK_TRUE`, the “Id” operand of `OpGroupNonUniformBroadcast` **can** be dynamically uniform within a subgroup, and the “Index” operand of `OpGroupNonUniformQuadBroadcast` **can** be dynamically uniform within the derivative group. If it is `VK_FALSE`, these operands **must** be constants.

If the `VkPhysicalDeviceVulkan12Features` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceVulkan12Features`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceVulkan12Features-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_2_FEATURES`

The `VkPhysicalDeviceVariablePointersFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceVariablePointersFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           variablePointersStorageBuffer;
    VkBool32           variablePointers;
} VkPhysicalDeviceVariablePointersFeatures;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `variablePointersStorageBuffer` specifies whether the implementation supports the SPIR-V `VariablePointersStorageBuffer` capability. When this feature is not enabled, shader modules **must** not declare the `SPV_KHR_variable_pointers` extension or the `VariablePointersStorageBuffer` capability.
- `variablePointers` specifies whether the implementation supports the SPIR-V `VariablePointers` capability. When this feature is not enabled, shader modules **must** not declare the

`VariablePointers` capability.

If the `VkPhysicalDeviceVariablePointersFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceVariablePointersFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage

- VUID-VkPhysicalDeviceVariablePointersFeatures-variablePointers-01431
If `variablePointers` is enabled then `variablePointersStorageBuffer` **must** also be enabled

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVariablePointersFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTERS_FEATURES`

The `VkPhysicalDeviceMultiviewFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceMultiviewFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           multiview;
    VkBool32           multiviewGeometryShader;
    VkBool32           multiviewTessellationShader;
} VkPhysicalDeviceMultiviewFeatures;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `multiview` specifies whether the implementation supports multiview rendering within a render pass. If this feature is not enabled, the view mask of each subpass **must** always be zero.
- `multiviewGeometryShader` specifies whether the implementation supports multiview rendering within a render pass, with `geometry shaders`. If this feature is not enabled, then a pipeline compiled against a subpass with a non-zero view mask **must** not include a geometry shader.
- `multiviewTessellationShader` specifies whether the implementation supports multiview rendering within a render pass, with `tessellation shaders`. If this feature is not enabled, then a pipeline compiled against a subpass with a non-zero view mask **must** not include any tessellation shaders.

If the `VkPhysicalDeviceMultiviewFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceMultiviewFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage

- VUID-VkPhysicalDeviceMultiviewFeatures-multiviewGeometryShader-00580
If `multiviewGeometryShader` is enabled then `multiview` **must** also be enabled
- VUID-VkPhysicalDeviceMultiviewFeatures-multiviewTessellationShader-00581
If `multiviewTessellationShader` is enabled then `multiview` **must** also be enabled

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceMultiviewFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_FEATURES`

The `VkPhysicalDeviceShaderAtomicFloatFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_shader_atomic_float
typedef struct VkPhysicalDeviceShaderAtomicFloatFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderBufferFloat32Atomics;
    VkBool32           shaderBufferFloat32AtomicAdd;
    VkBool32           shaderBufferFloat64Atomics;
    VkBool32           shaderBufferFloat64AtomicAdd;
    VkBool32           shaderSharedFloat32Atomics;
    VkBool32           shaderSharedFloat32AtomicAdd;
    VkBool32           shaderSharedFloat64Atomics;
    VkBool32           shaderSharedFloat64AtomicAdd;
    VkBool32           shaderImageFloat32Atomics;
    VkBool32           shaderImageFloat32AtomicAdd;
    VkBool32           sparseImageFloat32Atomics;
    VkBool32           sparseImageFloat32AtomicAdd;
} VkPhysicalDeviceShaderAtomicFloatFeaturesEXT;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderBufferFloat32Atomics` indicates whether shaders **can** perform 32-bit floating-point load, store and exchange atomic operations on storage buffers.

- `shaderBufferFloat32AtomicAdd` indicates whether shaders **can** perform 32-bit floating-point add atomic operations on storage buffers.
- `shaderBufferFloat64Atomics` indicates whether shaders **can** perform 64-bit floating-point load, store and exchange atomic operations on storage buffers.
- `shaderBufferFloat64AtomicAdd` indicates whether shaders **can** perform 64-bit floating-point add atomic operations on storage buffers.
- `shaderSharedFloat32Atomics` indicates whether shaders **can** perform 32-bit floating-point load, store and exchange atomic operations on shared memory.
- `shaderSharedFloat32AtomicAdd` indicates whether shaders **can** perform 32-bit floating-point add atomic operations on shared memory.
- `shaderSharedFloat64Atomics` indicates whether shaders **can** perform 64-bit floating-point load, store and exchange atomic operations on shared memory.
- `shaderSharedFloat64AtomicAdd` indicates whether shaders **can** perform 64-bit floating-point add atomic operations on shared memory.
- `shaderImageFloat32Atomics` indicates whether shaders **can** perform 32-bit floating-point load, store and exchange atomic image operations.
- `shaderImageFloat32AtomicAdd` indicates whether shaders **can** perform 32-bit floating-point add atomic image operations.
- `sparseImageFloat32Atomics` indicates whether 32-bit floating-point load, store and exchange atomic operations **can** be used on sparse images.
- `sparseImageFloat32AtomicAdd` indicates whether 32-bit floating-point add atomic operations **can** be used on sparse images.

If the `VkPhysicalDeviceShaderAtomicFloatFeaturesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceShaderAtomicFloatFeaturesEXT`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceShaderAtomicFloatFeaturesEXT-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_FLOAT_FEATURES_EXT`

The `VkPhysicalDeviceShaderAtomicInt64Features` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceShaderAtomicInt64Features {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderBufferInt64Atomics;
    VkBool32           shaderSharedInt64Atomics;
} VkPhysicalDeviceShaderAtomicInt64Features;
```

This structure describes the following features:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **shaderBufferInt64Atomics** indicates whether shaders **can** perform 64-bit unsigned and signed integer atomic operations on buffers.
- **shaderSharedInt64Atomics** indicates whether shaders **can** perform 64-bit unsigned and signed integer atomic operations on shared memory.

If the [VkPhysicalDeviceShaderAtomicInt64Features](#) structure is included in the **pNext** chain of the [VkPhysicalDeviceFeatures2](#) structure passed to [vkGetPhysicalDeviceFeatures2](#), it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a [VkDevice](#) with any features described by [VkPhysicalDeviceShaderAtomicInt64Features](#), it **must** add an instance of the structure, with the desired feature members set to **VK_TRUE**, to the **pNext** chain of [VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderAtomicInt64Features-sType-sType
sType must be **VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_INT64_FEATURES**

The [VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT](#) structure is defined as:

```
// Provided by VK_EXT_shader_image_atomic_int64
typedef struct VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderImageInt64Atomics;
    VkBool32           sparseImageInt64Atomics;
} VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT;
```

This structure describes the following features:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **shaderImageInt64Atomics** indicates whether shaders **can** support 64-bit unsigned and signed

integer atomic operations on images.

- `sparseImageInt64Atomics` indicates whether 64-bit integer atomics **can** be used on sparse images.

If the `VkPhysicalDeviceShaderAtomicInt64FeaturesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceShaderAtomicInt64FeaturesEXT`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT-sType-sType **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_IMAGE_ATOMIC_INT64_FEATURES_EXT`

The `VkPhysicalDevice8BitStorageFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDevice8BitStorageFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           storageBuffer8BitAccess;
    VkBool32           uniformAndStorageBuffer8BitAccess;
    VkBool32           storagePushConstant8;
} VkPhysicalDevice8BitStorageFeatures;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `storageBuffer8BitAccess` indicates whether objects in the `StorageBuffer`, or `PhysicalStorageBuffer` storage class with the `Block` decoration **can** have 8-bit integer members. If this feature is not enabled, 8-bit integer members **must** not be used in such objects. This also indicates whether shader modules **can** declare the `StorageBuffer8BitAccess` capability.
- `uniformAndStorageBuffer8BitAccess` indicates whether objects in the `Uniform` storage class with the `Block` decoration **can** have 8-bit integer members. If this feature is not enabled, 8-bit integer members **must** not be used in such objects. This also indicates whether shader modules **can** declare the `UniformAndStorageBuffer8BitAccess` capability.
- `storagePushConstant8` indicates whether objects in the `PushConstant` storage class **can** have 8-bit integer members. If this feature is not enabled, 8-bit integer members **must** not be used in such objects. This also indicates whether shader modules **can** declare the `StoragePushConstant8` capability.

If the `VkPhysicalDevice8BitStorageFeatures` structure is included in the `pNext` chain of the

`VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDevice8BitStorageFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- `VUID-VkPhysicalDevice8BitStorageFeatures-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_8BIT_STORAGE_FEATURES`

The `VkPhysicalDevice16BitStorageFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDevice16BitStorageFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           storageBuffer16BitAccess;
    VkBool32           uniformAndStorageBuffer16BitAccess;
    VkBool32           storagePushConstant16;
    VkBool32           storageInputOutput16;
} VkPhysicalDevice16BitStorageFeatures;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `storageBuffer16BitAccess` specifies whether objects in the `StorageBuffer`, or `PhysicalStorageBuffer` storage class with the `Block` decoration **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or 16-bit floating-point members **must** not be used in such objects. This also specifies whether shader modules **can** declare the `StorageBuffer16BitAccess` capability.
- `uniformAndStorageBuffer16BitAccess` specifies whether objects in the `Uniform` storage class with the `Block` decoration **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or 16-bit floating-point members **must** not be used in such objects. This also specifies whether shader modules **can** declare the `UniformAndStorageBuffer16BitAccess` capability.
- `storagePushConstant16` specifies whether objects in the `PushConstant` storage class **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or floating-point members **must** not be used in such objects. This also specifies whether shader modules **can** declare the `StoragePushConstant16` capability.
- `storageInputOutput16` specifies whether objects in the `Input` and `Output` storage classes **can** have 16-bit integer and 16-bit floating-point members. If this feature is not enabled, 16-bit integer or 16-bit floating-point members **must** not be used in such objects. This also specifies whether

shader modules **can** declare the `StorageInputOutput16` capability.

If the `VkPhysicalDevice16BitStorageFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDevice16BitStorageFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDevice16BitStorageFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_16BIT_STORAGE_FEATURES`

The `VkPhysicalDeviceShaderFloat16Int8Features` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceShaderFloat16Int8Features {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderFloat16;
    VkBool32           shaderInt8;
} VkPhysicalDeviceShaderFloat16Int8Features;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderFloat16` indicates whether 16-bit floats (halves) are supported in shader code. This also indicates whether shader modules **can** declare the `Float16` capability. However, this only enables a subset of the storage classes that SPIR-V allows for the `Float16` SPIR-V capability: Declaring and using 16-bit floats in the `Private`, `Workgroup`, and `Function` storage classes is enabled, while declaring them in the interface storage classes (e.g., `UniformConstant`, `Uniform`, `StorageBuffer`, `Input`, `Output`, and `PushConstant`) is not enabled.
- `shaderInt8` indicates whether 8-bit integers (signed and unsigned) are supported in shader code. This also indicates whether shader modules **can** declare the `Int8` capability. However, this only enables a subset of the storage classes that SPIR-V allows for the `Int8` SPIR-V capability: Declaring and using 8-bit integers in the `Private`, `Workgroup`, and `Function` storage classes is enabled, while declaring them in the interface storage classes (e.g., `UniformConstant`, `Uniform`, `StorageBuffer`, `Input`, `Output`, and `PushConstant`) is not enabled.

If the `VkPhysicalDeviceShaderFloat16Int8Features` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceShaderFloat16Int8Features`, it **must** add

an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderFloat16Int8Features-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_FLOAT16_INT8_FEATURES`

The `VkPhysicalDeviceShaderClockFeaturesKHR` structure is defined as:

```
// Provided by VK_KHR_shader_clock
typedef struct VkPhysicalDeviceShaderClockFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderSubgroupClock;
    VkBool32           shaderDeviceClock;
} VkPhysicalDeviceShaderClockFeaturesKHR;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderSubgroupClock` indicates whether shaders **can** perform `Subgroup` scoped clock reads.
- `shaderDeviceClock` indicates whether shaders **can** perform `Device` scoped clock reads.

If the `VkPhysicalDeviceShaderClockFeaturesKHR` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceShaderClockFeaturesKHR`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderClockFeaturesKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CLOCK_FEATURES_KHR`

The `VkPhysicalDeviceSamplerYcbcrConversionFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceSamplerYcbcrConversionFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           samplerYcbcrConversion;
} VkPhysicalDeviceSamplerYcbcrConversionFeatures;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `samplerYcbcrConversion` specifies whether the implementation supports `sampler Y'CBCR conversion`. If `samplerYcbcrConversion` is `VK_FALSE`, sampler Y'C_BC_R conversion is not supported, and samplers using sampler Y'C_BC_R conversion **must** not be used.

If the `VkPhysicalDeviceSamplerYcbcrConversionFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceSamplerYcbcrConversionFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSamplerYcbcrConversionFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_YCBCR_CONVERSION_FEATURES`

The `VkPhysicalDeviceProtectedMemoryFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceProtectedMemoryFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           protectedMemory;
} VkPhysicalDeviceProtectedMemoryFeatures;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `protectedMemory` specifies whether `protected memory` is supported.

If the `VkPhysicalDeviceProtectedMemoryFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to

indicate whether each corresponding feature is supported. If the application wishes to use a [VkDevice](#) with any features described by [VkPhysicalDeviceProtectedMemoryFeatures](#), it **must** add an instance of the structure, with the desired feature members set to [VK_TRUE](#), to the [pNext](#) chain of [VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceProtectedMemoryFeatures-sType-sType
[sType](#) **must** be [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROTECTED_MEMORY_FEATURES](#)

The [VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT](#) structure is defined as:

```
// Provided by VK_EXT_blend_operation_advanced
typedef struct VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           advancedBlendCoherentOperations;
} VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT;
```

This structure describes the following feature:

- [sType](#) is a [VkStructureType](#) value identifying this structure.
- [pNext](#) is [NULL](#) or a pointer to a structure extending this structure.
- [advancedBlendCoherentOperations](#) specifies whether blending using [advanced blend operations](#) is guaranteed to execute atomically and in [primitive order](#). If this is [VK_TRUE](#), [VK_ACCESS_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT](#) is treated the same as [VK_ACCESS_COLOR_ATTACHMENT_READ_BIT](#), and advanced blending needs no additional synchronization over basic blending. If this is [VK_FALSE](#), then memory dependencies are required to guarantee order between two advanced blending operations that occur on the same sample.

If the [VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT](#) structure is included in the [pNext](#) chain of the [VkPhysicalDeviceFeatures2](#) structure passed to [vkGetPhysicalDeviceFeatures2](#), it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a [VkDevice](#) with any features described by [VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT](#), it **must** add an instance of the structure, with the desired feature members set to [VK_TRUE](#), to the [pNext](#) chain of [VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT-sType-sType
[sType](#) **must** be [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BLEND_OPERATION_ADVANCED_FEATURES_EXT](#)

The [VkPhysicalDeviceShaderDrawParametersFeatures](#) structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceShaderDrawParametersFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderDrawParameters;
} VkPhysicalDeviceShaderDrawParametersFeatures;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderDrawParameters` specifies whether the implementation supports the SPIR-V `DrawParameters` capability. When this feature is not enabled, shader modules **must** not declare the `SPV_KHR_shader_draw_parameters` extension or the `DrawParameters` capability.

If the `VkPhysicalDeviceShaderDrawParametersFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceShaderDrawParametersFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderDrawParametersFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DRAW_PARAMETERS_FEATURES`

The `VkPhysicalDeviceDescriptorIndexingFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceDescriptorIndexingFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderInputAttachmentArrayDynamicIndexing;
    VkBool32           shaderUniformTexelBufferArrayDynamicIndexing;
    VkBool32           shaderStorageTexelBufferArrayDynamicIndexing;
    VkBool32           shaderUniformBufferArrayNonUniformIndexing;
    VkBool32           shaderSampledImageArrayNonUniformIndexing;
    VkBool32           shaderStorageBufferArrayNonUniformIndexing;
    VkBool32           shaderStorageImageArrayNonUniformIndexing;
    VkBool32           shaderInputAttachmentArrayNonUniformIndexing;
    VkBool32           shaderUniformTexelBufferArrayNonUniformIndexing;
    VkBool32           shaderStorageTexelBufferArrayNonUniformIndexing;
    VkBool32           descriptorBindingUniformBufferUpdateAfterBind;
    VkBool32           descriptorBindingSampledImageUpdateAfterBind;
    VkBool32           descriptorBindingStorageImageUpdateAfterBind;
    VkBool32           descriptorBindingStorageBufferUpdateAfterBind;
    VkBool32           descriptorBindingUniformTexelBufferUpdateAfterBind;
    VkBool32           descriptorBindingStorageTexelBufferUpdateAfterBind;
    VkBool32           descriptorBindingUpdateUnusedWhilePending;
    VkBool32           descriptorBindingPartiallyBound;
    VkBool32           descriptorBindingVariableDescriptorCount;
    VkBool32           runtimeDescriptorArray;
} VkPhysicalDeviceDescriptorIndexingFeatures;
```

This structure describes the following features:

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **shaderInputAttachmentArrayDynamicIndexing** indicates whether arrays of input attachments **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of **VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT** **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the **InputAttachmentArrayDynamicIndexing** capability.
- **shaderUniformTexelBufferArrayDynamicIndexing** indicates whether arrays of uniform texel buffers **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of **VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER** **must** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the **UniformTexelBufferArrayDynamicIndexing** capability.
- **shaderStorageTexelBufferArrayDynamicIndexing** indicates whether arrays of storage texel buffers **can** be indexed by integer expressions that are dynamically uniform within either the subgroup or the invocation group in shader code. If this feature is not enabled, resources with a descriptor type of **VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER** **must** be indexed only by constant

integral expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `StorageTexelBufferArrayDynamicIndexing` capability.

- `shaderUniformBufferArrayNonUniformIndexing` indicates whether arrays of uniform buffers **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `UniformBufferArrayNonUniformIndexing` capability.
- `shaderSampledImageArrayNonUniformIndexing` indicates whether arrays of samplers or sampled images **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_SAMPLER`, `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, or `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `SampledImageArrayNonUniformIndexing` capability.
- `shaderStorageBufferArrayNonUniformIndexing` indicates whether arrays of storage buffers **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `StorageBufferArrayNonUniformIndexing` capability.
- `shaderStorageImageArrayNonUniformIndexing` indicates whether arrays of storage images **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `StorageImageArrayNonUniformIndexing` capability.
- `shaderInputAttachmentArrayNonUniformIndexing` indicates whether arrays of input attachments **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `InputAttachmentArrayNonUniformIndexing` capability.
- `shaderUniformTexelBufferArrayNonUniformIndexing` indicates whether arrays of uniform texel buffers **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `UniformTexelBufferArrayNonUniformIndexing` capability.
- `shaderStorageTexelBufferArrayNonUniformIndexing` indicates whether arrays of storage texel buffers **can** be indexed by non-uniform integer expressions in shader code. If this feature is not enabled, resources with a descriptor type of `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` **must** not be indexed by non-uniform integer expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the `StorageTexelBufferArrayNonUniformIndexing` capability.

- `descriptorBindingUniformBufferUpdateAfterBind` indicates whether the implementation supports updating uniform buffer descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER`.
- `descriptorBindingSampledImageUpdateAfterBind` indicates whether the implementation supports updating sampled image descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_SAMPLER`, `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, or `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`.
- `descriptorBindingStorageImageUpdateAfterBind` indicates whether the implementation supports updating storage image descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`.
- `descriptorBindingStorageBufferUpdateAfterBind` indicates whether the implementation supports updating storage buffer descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER`.
- `descriptorBindingUniformTexelBufferUpdateAfterBind` indicates whether the implementation supports updating uniform texel buffer descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER`.
- `descriptorBindingStorageTexelBufferUpdateAfterBind` indicates whether the implementation supports updating storage texel buffer descriptors after a set is bound. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT` **must** not be used with `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`.
- `descriptorBindingUpdateUnusedWhilePending` indicates whether the implementation supports updating descriptors while the set is in use. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT` **must** not be used.
- `descriptorBindingPartiallyBound` indicates whether the implementation supports statically using a descriptor set binding in which some descriptors are not valid. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT` **must** not be used.
- `descriptorBindingVariableDescriptorCount` indicates whether the implementation supports descriptor sets with a variable-sized last binding. If this feature is not enabled, `VK_DESCRIPTOR_BINDING_VARIABLE_DESCRIPTOR_COUNT_BIT` **must** not be used.
- `runtimeDescriptorArray` indicates whether the implementation supports the SPIR-V `RuntimeDescriptorArray` capability. If this feature is not enabled, descriptors **must** not be declared in runtime arrays.

If the `VkPhysicalDeviceDescriptorIndexingFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceDescriptorIndexingFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDescriptorIndexingFeatures-sType-sType
sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_FEATURES

The `VkPhysicalDeviceVertexAttributeDivisorFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceVertexAttributeDivisorFeatures {  
    VkStructureType    sType;  
    void*              pNext;  
    VkBool32           vertexAttributeInstanceRateDivisor;  
    VkBool32           vertexAttributeInstanceRateZeroDivisor;  
} VkPhysicalDeviceVertexAttributeDivisorFeatures;
```

```
// Provided by VK_KHR_vertex_attribute_divisor  
typedef VkPhysicalDeviceVertexAttributeDivisorFeatures  
VkPhysicalDeviceVertexAttributeDivisorFeaturesKHR;
```

This structure describes the following features:

- sType is a `VkStructureType` value identifying this structure.
- pNext is `NULL` or a pointer to a structure extending this structure.
- vertexAttributeInstanceRateDivisor specifies whether vertex attribute fetching may be repeated in the case of instanced rendering.
- vertexAttributeInstanceRateZeroDivisor specifies whether a zero value for `VkVertexInputBindingDivisorDescriptionEXT::divisor` is supported.

If the `VkPhysicalDeviceVertexAttributeDivisorFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceVertexAttributeDivisorFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVertexAttributeDivisorFeatures-sType-sType
sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_FEATURES

The `VkPhysicalDeviceASTCDecodeFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_astc_decode_mode
typedef struct VkPhysicalDeviceASTCDecodeFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           decodeModeSharedExponent;
} VkPhysicalDeviceASTCDecodeFeaturesEXT;
```

This structure describes the following feature:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **decodeModeSharedExponent** indicates whether the implementation supports decoding ASTC compressed formats to **VK_FORMAT_E5B9G9R9_UFLOAT_PACK32** internal precision.

If the [VkPhysicalDeviceASTCDecodeFeaturesEXT](#) structure is included in the **pNext** chain of the [VkPhysicalDeviceFeatures2](#) structure passed to [vkGetPhysicalDeviceFeatures2](#), it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a [VkDevice](#) with any features described by [VkPhysicalDeviceASTCDecodeFeaturesEXT](#), it **must** add an instance of the structure, with the desired feature members set to **VK_TRUE**, to the **pNext** chain of [VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceASTCDecodeFeaturesEXT-sType-sType
sType must be **VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ASTC_DECODE_FEATURES_EXT**

The [VkPhysicalDeviceVulkanMemoryModelFeatures](#) structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceVulkanMemoryModelFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           vulkanMemoryModel;
    VkBool32           vulkanMemoryModelDeviceScope;
    VkBool32           vulkanMemoryModelAvailabilityVisibilityChains;
} VkPhysicalDeviceVulkanMemoryModelFeatures;
```

This structure describes the following features:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **vulkanMemoryModel** indicates whether shader modules **can** declare the [VulkanMemoryModel](#) capability.
- **vulkanMemoryModelDeviceScope** indicates whether the Vulkan Memory Model can use [Device](#)

scope synchronization. This also indicates whether shader modules **can** declare the `VulkanMemoryModelDeviceScope` capability.

- `vulkanMemoryModelAvailabilityVisibilityChains` indicates whether the Vulkan Memory Model can use [availability and visibility chains](#) with more than one element.

If the `VkPhysicalDeviceVulkanMemoryModelFeaturesKHR` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceVulkanMemoryModelFeaturesKHR`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVulkanMemoryModelFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_MEMORY_MODEL_FEATURES`

The `VkPhysicalDeviceScalarBlockLayoutFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceScalarBlockLayoutFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           scalarBlockLayout;
} VkPhysicalDeviceScalarBlockLayoutFeatures;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `scalarBlockLayout` indicates that the implementation supports the layout of resource blocks in shaders using [scalar alignment](#).

If the `VkPhysicalDeviceScalarBlockLayoutFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceScalarBlockLayoutFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceScalarBlockLayoutFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCALAR_BLOCK_LAYOUT_FEATURES`

The `VkPhysicalDeviceUniformBufferStandardLayoutFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceUniformBufferStandardLayoutFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           uniformBufferStandardLayout;
} VkPhysicalDeviceUniformBufferStandardLayoutFeatures;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `uniformBufferStandardLayout` indicates that the implementation supports the same layouts for uniform buffers as for storage and other kinds of buffers. See [Standard Buffer Layout](#).

If the `VkPhysicalDeviceUniformBufferStandardLayoutFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceUniformBufferStandardLayoutFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceUniformBufferStandardLayoutFeatures-sType-sType **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_UNIFORM_BUFFER_STANDARD_LAYOUT_FEATURES`

The `VkPhysicalDeviceDepthClipEnableFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_depth_clip_enable
typedef struct VkPhysicalDeviceDepthClipEnableFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           depthClipEnable;
} VkPhysicalDeviceDepthClipEnableFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `depthClipEnable` indicates that the implementation supports setting the depth clipping operation explicitly via the `VkPipelineRasterizationDepthClipStateCreateInfoEXT` pipeline state. Otherwise

depth clipping is only enabled when `VkPipelineRasterizationStateCreateInfo::depthClampEnable` is `VK_FALSE`.

If the `VkPhysicalDeviceDepthClipEnableFeaturesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceDepthClipEnableFeaturesEXT`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceDepthClipEnableFeaturesEXT-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_CLIP_ENABLE_FEATURES_EXT`

The `VkPhysicalDeviceBufferDeviceAddressFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceBufferDeviceAddressFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           bufferDeviceAddress;
    VkBool32           bufferDeviceAddressCaptureReplay;
    VkBool32           bufferDeviceAddressMultiDevice;
} VkPhysicalDeviceBufferDeviceAddressFeatures;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `bufferDeviceAddress` indicates that the implementation supports accessing buffer memory in shaders as storage buffers via an address queried from `vkGetBufferDeviceAddress`.
- `bufferDeviceAddressCaptureReplay` indicates that the implementation supports saving and reusing buffer and device addresses, e.g. for trace capture and replay.
- `bufferDeviceAddressMultiDevice` indicates that the implementation supports the `bufferDeviceAddress` feature for logical devices created with multiple physical devices. If this feature is not supported, buffer addresses **must** not be queried on a logical device created with more than one physical device.

NOTE

`bufferDeviceAddressMultiDevice` exists to allow certain legacy platforms to be able to support `bufferDeviceAddress` without needing to support shared GPU virtual addresses for multi-device configurations.

See `vkGetBufferDeviceAddress` for more information.

If the `VkPhysicalDeviceBufferDeviceAddressFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceBufferDeviceAddressFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceBufferDeviceAddressFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES`

The `VkPhysicalDeviceImagelessFramebufferFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceImagelessFramebufferFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           imagelessFramebuffer;
} VkPhysicalDeviceImagelessFramebufferFeatures;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `imagelessFramebuffer` indicates that the implementation supports specifying the image view for attachments at render pass begin time via `VkRenderPassAttachmentBeginInfo`.

If the `VkPhysicalDeviceImagelessFramebufferFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceImagelessFramebufferFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImagelessFramebufferFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGELESS_FRAMEBUFFER_FEATURES`

The `VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT` structure is defined as:


```
// Provided by VK_EXT_fragment_shader_interlock
typedef struct VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           fragmentShaderSampleInterlock;
    VkBool32           fragmentShaderPixelInterlock;
    VkBool32           fragmentShaderShadingRateInterlock;
} VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `fragmentShaderSampleInterlock` indicates that the implementation supports the `FragmentShaderSampleInterlockEXT` SPIR-V capability.
- `fragmentShaderPixelInterlock` indicates that the implementation supports the `FragmentShaderPixelInterlockEXT` SPIR-V capability.
- `fragmentShaderShadingRateInterlock` indicates that the implementation supports the `FragmentShaderShadingRateInterlockEXT` SPIR-V capability.

If the `VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- `VOID-VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_INTERLOCK_FEATURES_EXT`

The `VkPhysicalDeviceYcbcrImageArraysFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_ycbcr_image_arrays
typedef struct VkPhysicalDeviceYcbcrImageArraysFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           ycbcrImageArrays;
} VkPhysicalDeviceYcbcrImageArraysFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.

- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `ycbcrImageArrays` indicates that the implementation supports creating images with a format that requires `Y'CBCR conversion` and has multiple array layers.

If the `VkPhysicalDeviceYcbcrImageArraysFeaturesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceYcbcrImageArraysFeaturesEXT`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceYcbcrImageArraysFeaturesEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_IMAGE_ARRAYS_FEATURES_EXT`

The `VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderSubgroupExtendedTypes;
} VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderSubgroupExtendedTypes` is a boolean specifying whether subgroup operations can use 8-bit integer, 16-bit integer, 64-bit integer, 16-bit floating-point, and vectors of these types in `group operations` with `subgroup scope`, if the implementation supports the types.

If the `VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_EXTENDED_TYPES_FEATURES`

The `VkPhysicalDeviceHostQueryResetFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceHostQueryResetFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           hostQueryReset;
} VkPhysicalDeviceHostQueryResetFeatures;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `hostQueryReset` indicates that the implementation supports resetting queries from the host with `vkResetQueryPool`.

If the `VkPhysicalDeviceHostQueryResetFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceHostQueryResetFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceHostQueryResetFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_QUERY_RESET_FEATURES`

The `VkPhysicalDeviceTimelineSemaphoreFeatures` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceTimelineSemaphoreFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           timelineSemaphore;
} VkPhysicalDeviceTimelineSemaphoreFeatures;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `timelineSemaphore` indicates whether semaphores created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE` are supported.

If the `VkPhysicalDeviceTimelineSemaphoreFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceTimelineSemaphoreFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceTimelineSemaphoreFeatures-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_FEATURES`

The `VkPhysicalDeviceExternalSciSyncFeaturesNV` structure is defined as:

```
// Provided by VK_NV_external_sci_sync
typedef struct VkPhysicalDeviceExternalSciSyncFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           sciSyncFence;
    VkBool32           sciSyncSemaphore;
    VkBool32           sciSyncImport;
    VkBool32           sciSyncExport;
} VkPhysicalDeviceExternalSciSyncFeaturesNV;
```

The members of the `VkPhysicalDeviceExternalSciSyncFeaturesNV` structure describe the following features:

- `sciSyncFence` indicates whether external fences created with a handle type of `VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV` and `VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV` are supported for import and/or export.
- `sciSyncSemaphore` indicates whether external semaphores created with a handle type of `VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV` are supported for import and/or export.
- `sciSyncImport` indicates whether `NvSciSyncObj` import functionality is supported. If `sciSyncImport` is `VK_TRUE`, `VkFence` and/or `VkSemaphore` support importing `NvSciSyncObj` from applications. In this case, the application is responsible for the resource management of the `NvSciSyncObj`.
- `sciSyncExport` indicates whether `NvSciSyncObj` export functionality is supported. If `sciSyncExport` is `VK_TRUE`, `VkFence` and/or `VkSemaphore` support exporting `NvSciSyncObj` created by the driver to applications. In this case, the driver is responsible for the resource management of the `NvSciSyncObj`.

Table 44. Functionality Supported for `NvSciSync` Features

Features	<code>sciSyncImport</code>	<code>sciSyncExport</code>	Always supported ¹

sciSyncFence	vkImportFenceSciSyncFenceNV, vkImportFenceSciSyncObjNV	VkExportFenceSciSyncInfoNV	vkGetFenceSciSyncFenceNV, vkGetFenceSciSyncObjNV, vkGetPhysicalDeviceSciSyncAttributesNV (with VK_SCI_SYNC_PRIMITIVE_TYPE_FENCE_NV)
sciSyncSemaphore	vkImportSemaphoreSciSyncObjNV	VkExportSemaphoreSciSyncInfoNV	vkGetSemaphoreSciSyncObjNV, vkGetPhysicalDeviceSciSyncAttributesNV (with VK_SCI_SYNC_PRIMITIVE_TYPE_SEMAPHORE_NV)

1

Functionality in this column is always available.

The [Functionality supported for NvSciSync features](#) table summarizes the functionality enabled by the [VkPhysicalDeviceExternalSciSyncFeaturesNV](#) structure. There are two orthogonal pieces of functionality: fence and semaphore support; import and export support. Each entry in the body of the table summarizes the functionality that **can** be used when the given features are supported and enabled. This summarizes Valid Usage statements that are added elsewhere in this specification.

If the [VkPhysicalDeviceExternalSciSyncFeaturesNV](#) structure is included in the [pNext](#) chain of the [VkPhysicalDeviceFeatures2](#) structure passed to [vkGetPhysicalDeviceFeatures2](#), it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a [VkDevice](#) with any features described by [VkPhysicalDeviceExternalSciSyncFeaturesNV](#), it **must** add an instance of the structure, with the desired feature members set to [VK_TRUE](#), to the [pNext](#) chain of [VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalSciSyncFeaturesNV-sType-sType
sType **must** be [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SCI_SYNC_FEATURES_NV](#)

The [VkPhysicalDeviceExternalSciSync2FeaturesNV](#) structure is defined as:

```
// Provided by VK_NV_external_sci_sync2
typedef struct VkPhysicalDeviceExternalSciSync2FeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           sciSyncFence;
    VkBool32           sciSyncSemaphore2;
    VkBool32           sciSyncImport;
    VkBool32           sciSyncExport;
} VkPhysicalDeviceExternalSciSync2FeaturesNV;
```

The members of the `VkPhysicalDeviceExternalSciSync2FeaturesNV` structure describe the following features:

- `sciSyncFence` indicates whether external fences created with a handle type of `VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV` and `VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV` are supported for import and/or export.
- `sciSyncSemaphore2` indicates whether semaphore SciSync pools are supported and semaphores can be created from `NvSciSyncObj` via `VkSemaphoreSciSyncPoolNV` objects. In this case, the application is responsible for the resource management of the `NvSciSyncObj`.
- `sciSyncImport` indicates whether `NvSciSyncObj` import functionality is supported. If `sciSyncImport` is `VK_TRUE`, `VkFence` and/or `VkSemaphore` support importing `NvSciSyncObj` from applications. In this case, the application is responsible for the resource management of the `NvSciSyncObj`.
- `sciSyncExport` indicates whether `NvSciSyncObj` export functionality is supported. If `sciSyncExport` is `VK_TRUE`, `VkFence` supports exporting `NvSciSyncObj` created by the driver to applications. In this case, the driver is responsible for the resource management of the `NvSciSyncObj`.

Table 45. Functionality Supported for NvSciSync Features

Features	<code>sciSyncImport</code>	<code>sciSyncExport</code>	Always supported ¹
<code>sciSyncFence</code>	<code>vkImportFenceSciSyncFenceNV</code> , <code>vkImportFenceSciSyncObjNV</code>	<code>VkExportFenceSciSyncInfoNV</code>	<code>vkGetFenceSciSyncFenceNV</code> , <code>vkGetFenceSciSyncObjNV</code> , <code>vkGetPhysicalDeviceSciSyncAttributesNV</code> (with <code>VK_SCI_SYNC_PRIMITIVE_TYPE_FENCE_NV</code>)
<code>sciSyncSemaphore2</code>	<code>vkCreateSemaphoreSciSyncPoolNV</code> , <code>VkSemaphoreSciSyncCreateInfoNV</code>	n/a	<code>vkGetPhysicalDeviceSciSyncAttributesNV</code> (with <code>VK_SCI_SYNC_PRIMITIVE_TYPE_SEMAPHORE_NV</code>)

1

Functionality in this column is always available.

The [Functionality supported for NvSciSync features](#) table summarizes the functionality enabled by

the `VkPhysicalDeviceExternalSciSync2FeaturesNV` structure. There are two orthogonal pieces of functionality: fence and semaphore support; import and export support. Each entry in the body of the table summarizes the functionality that **can** be used when the given features are supported and enabled. This summarizes Valid Usage statements that are added elsewhere in this specification.

If the `VkPhysicalDeviceExternalSciSync2FeaturesNV` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceExternalSciSync2FeaturesNV`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalSciSync2FeaturesNV-sType-sType
sType **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SCI_SYNC_2_FEATURES_NV`

The `VkPhysicalDeviceExternalMemorySciBufFeaturesNV` structure is defined as:

```
// Provided by VK_NV_external_memory_sci_buf
typedef struct VkPhysicalDeviceExternalMemorySciBufFeaturesNV {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           sciBufImport;
    VkBool32           sciBufExport;
} VkPhysicalDeviceExternalMemorySciBufFeaturesNV;
```

```
// Provided by VK_NV_external_memory_sci_buf
typedef VkPhysicalDeviceExternalMemorySciBufFeaturesNV
VkPhysicalDeviceExternalSciBufFeaturesNV;
```

The members of the `VkPhysicalDeviceExternalMemorySciBufFeaturesNV` structure describe the following features:

- `sciBufImport` indicates whether `NvSciBufObj` import functionality is supported. If `sciBufImport` is `VK_TRUE`, `VkDeviceMemory` supports importing `NvSciBufObj` from applications. In this case, the application is responsible for the resource management of the `NvSciBufObj`.
- `sciBufExport` indicates whether `NvSciBufObj` export functionality is supported. If `sciBufExport` is `VK_TRUE`, `VkDeviceMemory` supports exporting `NvSciBufObj` created by the driver to applications. In this case, the driver is responsible for the resource management of the `NvSciBufObj`.

Table 46. Functionality Supported for NvSciBuf Features

Features	Functionality

sciBufImport	VkImportMemorySciBufInfoNV, vkGetPhysicalDeviceExternalMemorySciBufPropertiesNV
sciBufExport	VkExportMemorySciBufInfoNV
Always supported ¹	vkGetPhysicalDeviceSciBufAttributesNV, vkGetMemorySciBufNV,

1

Functionality in this row is always available.

The [Functionality supported for NvSciBuf features](#) table summarizes the functionality enabled by the [VkPhysicalDeviceExternalMemorySciBufFeaturesNV](#) structure. Each entry in the body of the table summarizes the functionality that **can** be used when the given features are supported and enabled. This summarizes Valid Usage statements that are added elsewhere in this specification.

If the [VkPhysicalDeviceExternalMemorySciBufFeaturesNV](#) structure is included in the [pNext](#) chain of the [VkPhysicalDeviceFeatures2](#) structure passed to [vkGetPhysicalDeviceFeatures2](#), it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a [VkDevice](#) with any features described by [VkPhysicalDeviceExternalMemorySciBufFeaturesNV](#), it **must** add an instance of the structure, with the desired feature members set to [VK_TRUE](#), to the [pNext](#) chain of [VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalMemorySciBufFeaturesNV-sType-sType
[sType](#) **must** be [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_SCI_BUF_FEATURES_NV](#)

The [VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX](#) structure is defined as:

```
// Provided by VK_QNX_external_memory_screen_buffer
typedef struct VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           screenBufferImport;
} VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX;
```

The members of the [VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX](#) structure describe the following features:

- [screenBufferImport](#) indicates whether QNX Screen buffer import functionality is supported. If [screenBufferImport](#) is [VK_TRUE](#), [VkDeviceMemory](#) supports importing [_screen_buffer](#) from applications. In this case, the application is responsible for the resource management of the [_screen_buffer](#).

Table 47. Functionality Supported for QNX Screen Buffer Features

Features	Functionality
<code>screenBufferImport</code>	<code>VkImportScreenBufferInfoQNX</code>
Always supported ¹	<code>vkGetScreenBufferPropertiesQNX</code> , <code>VkScreenBufferPropertiesQNX</code> , <code>VkScreenBufferFormatPropertiesQNX</code> , <code>VkExternalFormatQNX</code>

1

Functionality in this row is always available.

The [Functionality supported for QNX Screen buffer features](#) table summarizes the functionality enabled by the `VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX` structure. Each entry in the body of the table summarizes the functionality that **can** be used when the given features are supported and enabled. This summarizes Valid Usage statements that are added elsewhere in this specification.

If the `VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX-sType-sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_SCREEN_BUFFER_FEATURES_QNX`

The `VkPhysicalDeviceIndexTypeUint8Features` structure is defined as:

```
typedef struct VkPhysicalDeviceIndexTypeUint8Features {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           indexTypeUint8;
} VkPhysicalDeviceIndexTypeUint8Features;
```

```
// Provided by VK_KHR_index_type_uint8
typedef VkPhysicalDeviceIndexTypeUint8Features
VkPhysicalDeviceIndexTypeUint8FeaturesKHR;
```

```
// Provided by VK_EXT_index_type_uint8
typedef VkPhysicalDeviceIndexTypeUint8Features
VkPhysicalDeviceIndexTypeUint8FeaturesEXT;
```

This structure describes the following feature:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **indexTypeUint8** indicates that **VK_INDEX_TYPE_UINT8** can be used with [vkCmdBindIndexBuffer](#).

If the [VkPhysicalDeviceIndexTypeUint8Features](#) structure is included in the **pNext** chain of the [VkPhysicalDeviceFeatures2](#) structure passed to [vkGetPhysicalDeviceFeatures2](#), it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a [VkDevice](#) with any features described by [VkPhysicalDeviceIndexTypeUint8Features](#), it **must** add an instance of the structure, with the desired feature members set to **VK_TRUE**, to the **pNext** chain of [VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceIndexTypeUint8Features-sType-sType
sType must be **VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES**

The [VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures](#) structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           separateDepthStencilLayouts;
} VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures;
```

This structure describes the following feature:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **separateDepthStencilLayouts** indicates whether the implementation supports a [VkImageMemoryBarrier](#) for a depth/stencil image with only one of **VK_IMAGE_ASPECT_DEPTH_BIT** or **VK_IMAGE_ASPECT_STENCIL_BIT** set, and whether **VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL**, **VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL**, **VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL**, or **VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL** can be used.

If the [VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures](#) structure is included in the **pNext** chain of the [VkPhysicalDeviceFeatures2](#) structure passed to [vkGetPhysicalDeviceFeatures2](#), it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a

[VkDevice](#) with any features described by [VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures](#), it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of [VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures-sType-sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SEPARATE_DEPTH_STENCIL_LAYOUTS_FEATURES`

The [VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures](#) structure is defined as:

```
typedef struct VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderDemoteToHelperInvocation;
} VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures;
```

```
// Provided by VK_EXT_shader_demote_to_helper_invocation
typedef VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures
VkPhysicalDeviceShaderDemoteToHelperInvocationFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a [VkStructureType](#) value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderDemoteToHelperInvocation` indicates whether the implementation supports the SPIR-V `DemoteToHelperInvocationEXT` capability.

If the [VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures](#) structure is included in the `pNext` chain of the [VkPhysicalDeviceFeatures2](#) structure passed to [vkGetPhysicalDeviceFeatures2](#), it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a [VkDevice](#) with any features described by [VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures](#), it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of [VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceShaderDemoteToHelperInvocationFeatures-sType-sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DEMOTE_TO_HELPER_INVOCATION_FEATURES`

The [VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT](#) structure is defined as:

```
// Provided by VK_EXT_texel_buffer_alignment
typedef struct VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           texelBufferAlignment;
} VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT;
```

This structure describes the following feature:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **texelBufferAlignment** indicates whether the implementation uses more specific alignment requirements advertised in [VkPhysicalDeviceTexelBufferAlignmentProperties](#) rather than [VkPhysicalDeviceLimits::minTexelBufferOffsetAlignment](#).

If the [VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT](#) structure is included in the **pNext** chain of the [VkPhysicalDeviceFeatures2](#) structure passed to [vkGetPhysicalDeviceFeatures2](#), it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a [VkDevice](#) with any features described by [VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT](#), it **must** add an instance of the structure, with the desired feature members set to **VK_TRUE**, to the **pNext** chain of [VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT-sType-sType
sType must be **VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_FEATURES_EXT**

The [VkPhysicalDeviceTextureCompressionASTCHDRFeatures](#) structure is defined as:

```
typedef struct VkPhysicalDeviceTextureCompressionASTCHDRFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           textureCompressionASTC_HDR;
} VkPhysicalDeviceTextureCompressionASTCHDRFeatures;
```

```
// Provided by VK_EXT_texture_compression_astc_hdr
typedef VkPhysicalDeviceTextureCompressionASTCHDRFeatures
VkPhysicalDeviceTextureCompressionASTCHDRFeaturesEXT;
```

This structure describes the following feature:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.

- `textureCompressionASTC_HDR` indicates whether all of the ASTC HDR compressed texture formats are supported. If this feature is enabled, then the `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`, `VK_FORMAT_FEATURE_BLIT_SRC_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` features **must** be supported in `optimalTilingFeatures` for the following formats:

- `VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK`
- `VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK`

To query for additional properties, or if the feature is not enabled, `vkGetPhysicalDeviceFormatProperties` and `vkGetPhysicalDeviceImageFormatProperties` **can** be used to check for supported properties of individual formats as normal.

If the `VkPhysicalDeviceTextureCompressionASTCHDRFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceTextureCompressionASTCHDRFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceTextureCompressionASTCHDRFeatures-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXTURE_COMPRESSION_ASTC_HDR_FEATURES`

The `VkPhysicalDeviceLineRasterizationFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceLineRasterizationFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32            rectangularLines;
    VkBool32            bresenhamLines;
    VkBool32            smoothLines;
    VkBool32            stippledRectangularLines;
    VkBool32            stippledBresenhamLines;
    VkBool32            stippledSmoothLines;
} VkPhysicalDeviceLineRasterizationFeatures;
```

```
// Provided by VK_KHR_line_rasterization
typedef VkPhysicalDeviceLineRasterizationFeatures
VkPhysicalDeviceLineRasterizationFeaturesKHR;
```

```
// Provided by VK_EXT_line_rasterization
typedef VkPhysicalDeviceLineRasterizationFeatures
VkPhysicalDeviceLineRasterizationFeaturesEXT;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `rectangularLines` indicates whether the implementation supports [rectangular line rasterization](#).
- `bresenhamLines` indicates whether the implementation supports [Bresenham-style line rasterization](#).
- `smoothLines` indicates whether the implementation supports [smooth line rasterization](#).
- `stippledRectangularLines` indicates whether the implementation supports [stippled line rasterization](#) with `VK_LINE_RASTERIZATION_MODE_RECTANGULAR` lines.
- `stippledBresenhamLines` indicates whether the implementation supports [stippled line rasterization](#) with `VK_LINE_RASTERIZATION_MODE_BRESENHAM` lines.
- `stippledSmoothLines` indicates whether the implementation supports [stippled line rasterization](#) with `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH` lines.

If the `VkPhysicalDeviceLineRasterizationFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceLineRasterizationFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceLineRasterizationFeatures-sType-sType
sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES

The `VkPhysicalDeviceSubgroupSizeControlFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceSubgroupSizeControlFeatures {  
    VkStructureType    sType;  
    void*              pNext;  
    VkBool32           subgroupSizeControl;  
    VkBool32           computeFullSubgroups;  
} VkPhysicalDeviceSubgroupSizeControlFeatures;
```

```
// Provided by VK_EXT_subgroup_size_control  
typedef VkPhysicalDeviceSubgroupSizeControlFeatures  
VkPhysicalDeviceSubgroupSizeControlFeaturesEXT;
```

This structure describes the following features:

- sType is a `VkStructureType` value identifying this structure.
- pNext is `NULL` or a pointer to a structure extending this structure.
- subgroupSizeControl indicates whether the implementation supports controlling shader subgroup sizes via the `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT` flag and the `VkPipelineShaderStageRequiredSubgroupSizeCreateInfo` structure.
- computeFullSubgroups indicates whether the implementation supports requiring full subgroups in compute shaders via the `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT` flag.

If the `VkPhysicalDeviceSubgroupSizeControlFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceSubgroupSizeControlFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSubgroupSizeControlFeatures-sType-sType
sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_FEATURES

The `VkPhysicalDeviceExtendedDynamicStateFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_extended_dynamic_state
typedef struct VkPhysicalDeviceExtendedDynamicStateFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           extendedDynamicState;
} VkPhysicalDeviceExtendedDynamicStateFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `extendedDynamicState` indicates that the implementation supports the following dynamic states:
 - `VK_DYNAMIC_STATE_CULL_MODE`
 - `VK_DYNAMIC_STATE_FRONT_FACE`
 - `VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY`
 - `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT`
 - `VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT`
 - `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE`
 - `VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE`
 - `VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE`
 - `VK_DYNAMIC_STATE_DEPTH_COMPARE_OP`
 - `VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE`
 - `VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE`
 - `VK_DYNAMIC_STATE_STENCIL_OP`

If the `VkPhysicalDeviceExtendedDynamicStateFeaturesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceExtendedDynamicStateFeaturesEXT`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExtendedDynamicStateFeaturesEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_FEATURES_EXT`

The `VkPhysicalDeviceExtendedDynamicState2FeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_extended_dynamic_state2
typedef struct VkPhysicalDeviceExtendedDynamicState2FeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           extendedDynamicState2;
    VkBool32           extendedDynamicState2LogicOp;
    VkBool32           extendedDynamicState2PatchControlPoints;
} VkPhysicalDeviceExtendedDynamicState2FeaturesEXT;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `extendedDynamicState2` indicates that the implementation supports the following dynamic states:
 - `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE`
 - `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE`
 - `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE`
- `extendedDynamicState2LogicOp` indicates that the implementation supports the following dynamic state:
 - `VK_DYNAMIC_STATE_LOGIC_OP_EXT`
- `extendedDynamicState2PatchControlPoints` indicates that the implementation supports the following dynamic state:
 - `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT`

If the `VkPhysicalDeviceExtendedDynamicState2FeaturesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceExtendedDynamicState2FeaturesEXT`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExtendedDynamicState2FeaturesEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_2_FEATURES_EXT`

The `VkPhysicalDeviceGlobalPriorityQueryFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceGlobalPriorityQueryFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           globalPriorityQuery;
} VkPhysicalDeviceGlobalPriorityQueryFeatures;
```

```
// Provided by VK_KHR_global_priority
typedef VkPhysicalDeviceGlobalPriorityQueryFeatures
VkPhysicalDeviceGlobalPriorityQueryFeaturesKHR;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `globalPriorityQuery` indicates whether the implementation supports the ability to query global queue priorities.

If the `VkPhysicalDeviceGlobalPriorityQueryFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceGlobalPriorityQueryFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceGlobalPriorityQueryFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GLOBAL_PRIORITY_QUERY_FEATURES`

The `VkPhysicalDeviceRobustness2FeaturesKHR` structure is defined as:

```
typedef struct VkPhysicalDeviceRobustness2FeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           robustBufferAccess2;
    VkBool32           robustImageAccess2;
    VkBool32           nullDescriptor;
} VkPhysicalDeviceRobustness2FeaturesKHR;
```

```
// Provided by VK_EXT_robustness2
typedef VkPhysicalDeviceRobustness2FeaturesKHR VkPhysicalDeviceRobustness2FeaturesEXT;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `robustBufferAccess2` enables `Robust Buffer Access 2` guarantees for shader buffer accesses.
- `robustImageAccess2` enables `Robust Image Access 2` guarantees for shader image accesses.
- `nullDescriptor` indicates whether descriptors **can** be written with a `VK_NULL_HANDLE` resource or view, which are considered valid to access and act as if the descriptor were bound to nothing.

If the `VkPhysicalDeviceRobustness2FeaturesKHR` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceRobustness2FeaturesKHR`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage

- VUID-VkPhysicalDeviceRobustness2FeaturesKHR-robustBufferAccess2-04000
If `robustBufferAccess2` is enabled then `robustBufferAccess` **must** also be enabled

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRobustness2FeaturesKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_FEATURES_KHR`

`nullDescriptor` support requires the `VK_KHR_robustness2` or `VK_EXT_robustness2` extension.

The `VkPhysicalDeviceImageRobustnessFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceImageRobustnessFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           robustImageAccess;
} VkPhysicalDeviceImageRobustnessFeatures;
```

```
// Provided by VK_EXT_image_robustness
typedef VkPhysicalDeviceImageRobustnessFeatures
VkPhysicalDeviceImageRobustnessFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- `robustImageAccess` enables [Robust Image Access](#) guarantees for shader image accesses.

If the `VkPhysicalDeviceImageRobustnessFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceImageRobustnessFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageRobustnessFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ROBUSTNESS_FEATURES`

The `VkPhysicalDeviceShaderTerminateInvocationFeatures` structure is defined as:

```
typedef struct VkPhysicalDeviceShaderTerminateInvocationFeatures {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderTerminateInvocation;
} VkPhysicalDeviceShaderTerminateInvocationFeatures;
```

```
// Provided by VK_KHR_shader_terminate_invocation
typedef VkPhysicalDeviceShaderTerminateInvocationFeatures
VkPhysicalDeviceShaderTerminateInvocationFeaturesKHR;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `shaderTerminateInvocation` specifies whether the implementation supports SPIR-V modules that use the `SPV_KHR_terminate_invocation` extension.

If the `VkPhysicalDeviceShaderTerminateInvocationFeatures` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceShaderTerminateInvocationFeatures`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceShaderTerminateInvocationFeatures-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TERMINATE_INVOCATION_FEATURES`

The `VkPhysicalDeviceCustomBorderColorFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_custom_border_color
typedef struct VkPhysicalDeviceCustomBorderColorFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           customBorderColors;
    VkBool32           customBorderColorWithoutFormat;
} VkPhysicalDeviceCustomBorderColorFeaturesEXT;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `customBorderColors` indicates that the implementation supports providing a `borderColor` value with one of the following values at sampler creation time:
 - `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT`
 - `VK_BORDER_COLOR_INT_CUSTOM_EXT`
- `customBorderColorWithoutFormat` indicates that explicit formats are not required for custom border colors and the value of the `format` member of the `VkSamplerCustomBorderColorCreateInfoEXT` structure **may** be `VK_FORMAT_UNDEFINED`. If this feature bit is not set, applications **must** provide the `VkFormat` of the image view(s) being sampled by this sampler in the `format` member of the `VkSamplerCustomBorderColorCreateInfoEXT` structure.

If the `VkPhysicalDeviceCustomBorderColorFeaturesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceCustomBorderColorFeaturesEXT`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCustomBorderColorFeaturesEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUSTOM_BORDER_COLOR_FEATURES_EXT`

The `VkPhysicalDeviceVulkanSC10Features` structure is defined as:


```
// Provided by VKSC_VERSION_1_0
typedef struct VkPhysicalDeviceVulkanSC10Features {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           shaderAtomicInstructions;
} VkPhysicalDeviceVulkanSC10Features;
```

This structure describes the following features:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **shaderAtomicInstructions** indicates whether this implementation supports shaders which use the SPIR-V *OpAtomic** instructions.

If the [VkPhysicalDeviceVulkanSC10Features](#) structure is included in the **pNext** chain of the [VkPhysicalDeviceFeatures2](#) structure passed to [vkGetPhysicalDeviceFeatures2](#), it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a [VkDevice](#) with any features described by [VkPhysicalDeviceVulkanSC10Features](#), it **must** add an instance of the structure, with the desired feature members set to **VK_TRUE**, to the **pNext** chain of [VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVulkanSC10Features-sType-sType
sType must be **VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_SC_1_0_FEATURES**

The [VkPhysicalDevicePerformanceQueryFeaturesKHR](#) structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkPhysicalDevicePerformanceQueryFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           performanceCounterQueryPools;
    VkBool32           performanceCounterMultipleQueryPools;
} VkPhysicalDevicePerformanceQueryFeaturesKHR;
```

This structure describes the following features:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **performanceCounterQueryPools** indicates whether the implementation supports performance counter query pools.
- **performanceCounterMultipleQueryPools** indicates whether the implementation supports using multiple performance query pools in a primary command buffer and secondary command

buffers executed within it.

If the `VkPhysicalDevicePerformanceQueryFeaturesKHR` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDevicePerformanceQueryFeaturesKHR`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDevicePerformanceQueryFeaturesKHR-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PERFORMANCE_QUERY_FEATURES_KHR`

The `VkPhysicalDevice4444FormatsFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_4444_formats
typedef struct VkPhysicalDevice4444FormatsFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           formatA4R4G4B4;
    VkBool32           formatA4B4G4R4;
} VkPhysicalDevice4444FormatsFeaturesEXT;
```

This structure describes the following features:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `formatA4R4G4B4` indicates that the implementation **must** support using a `VkFormat` of `VK_FORMAT_A4R4G4B4_UNORM_PACK16_EXT` with at least the following `VkFormatFeatureFlagBits`:
 - `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`
 - `VK_FORMAT_FEATURE_BLIT_SRC_BIT`
 - `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`
- `formatA4B4G4R4` indicates that the implementation **must** support using a `VkFormat` of `VK_FORMAT_A4B4G4R4_UNORM_PACK16_EXT` with at least the following `VkFormatFeatureFlagBits`:
 - `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`
 - `VK_FORMAT_FEATURE_BLIT_SRC_BIT`
 - `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT`

If the `VkPhysicalDevice4444FormatsFeaturesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDevice4444FormatsFeaturesEXT`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of

[VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- VUID-VkPhysicalDevice4444FormatsFeaturesEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_4444_FORMATS_FEATURES_EXT`

The `VkPhysicalDeviceSynchronization2Features` structure is defined as:

```
typedef struct VkPhysicalDeviceSynchronization2Features {  
    VkStructureType    sType;  
    void*              pNext;  
    VkBool32           synchronization2;  
} VkPhysicalDeviceSynchronization2Features;
```

```
// Provided by VK_KHR_synchronization2  
typedef VkPhysicalDeviceSynchronization2Features  
VkPhysicalDeviceSynchronization2FeaturesKHR;
```

This structure describes the following feature:

- `sType` is a [VkStructureType](#) value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `synchronization2` indicates whether the implementation supports the new set of synchronization commands introduced in [VK_KHR_synchronization2](#).

If the `VkPhysicalDeviceSynchronization2Features` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a [VkDevice](#) with any features described by `VkPhysicalDeviceSynchronization2Features`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the [VkDevice](#).

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSynchronization2Features-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SYNCHRONIZATION_2_FEATURES`

The `VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_vertex_input_dynamic_state
typedef struct VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           vertexInputDynamicState;
} VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT;
```

This structure describes the following feature:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **vertexInputDynamicState** indicates that the implementation supports the following dynamic states:
 - **VK_DYNAMIC_STATE_VERTEX_INPUT_EXT**

If the [VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT](#) structure is included in the **pNext** chain of the [VkPhysicalDeviceFeatures2](#) structure passed to [vkGetPhysicalDeviceFeatures2](#), it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a [VkDevice](#) with any features described by [VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT](#), it **must** add an instance of the structure, with the desired feature members set to **VK_TRUE**, to the **pNext** chain of [VkDeviceCreateInfo](#) when creating the [VkDevice](#).

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT-sType-sType **must** be **VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_INPUT_DYNAMIC_STATE_FEATURES_EXT**

The [VkPhysicalDeviceFragmentShadingRateFeaturesKHR](#) structure is defined as:

```
// Provided by VK_KHR_fragment_shading_rate
typedef struct VkPhysicalDeviceFragmentShadingRateFeaturesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           pipelineFragmentShadingRate;
    VkBool32           primitiveFragmentShadingRate;
    VkBool32           attachmentFragmentShadingRate;
} VkPhysicalDeviceFragmentShadingRateFeaturesKHR;
```

This structure describes the following features:

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **pipelineFragmentShadingRate** indicates that the implementation supports the [pipeline fragment](#)

shading rate.

- `primitiveFragmentShadingRate` indicates that the implementation supports the [primitive fragment shading rate](#).
- `attachmentFragmentShadingRate` indicates that the implementation supports the [attachment fragment shading rate](#).

If the `VkPhysicalDeviceFragmentShadingRateFeaturesKHR` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceFragmentShadingRateFeaturesKHR`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentShadingRateFeaturesKHR-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_FEATURES_KHR`

The `VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_ycbcr_2plane_444_formats
typedef struct VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           ycbcr2plane444Formats;
} VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `ycbcr2plane444Formats` indicates that the implementation supports the following 2-plane 444 Y'C_BR formats:
 - `VK_FORMAT_G8_B8R8_2PLANE_444_UNORM`
 - `VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16`
 - `VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16`
 - `VK_FORMAT_G16_B16R16_2PLANE_444_UNORM`

If the `VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT-sType-sType **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_2_PLANE_444_FORMATS_FEATURES_EXT`

The `VkPhysicalDeviceColorWriteEnableFeaturesEXT` structure is defined as:

```
// Provided by VK_EXT_color_write_enable
typedef struct VkPhysicalDeviceColorWriteEnableFeaturesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           colorWriteEnable;
} VkPhysicalDeviceColorWriteEnableFeaturesEXT;
```

This structure describes the following feature:

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `colorWriteEnable` indicates that the implementation supports the dynamic state `VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT`.

If the `VkPhysicalDeviceColorWriteEnableFeaturesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceFeatures2` structure passed to `vkGetPhysicalDeviceFeatures2`, it is filled in to indicate whether each corresponding feature is supported. If the application wishes to use a `VkDevice` with any features described by `VkPhysicalDeviceColorWriteEnableFeaturesEXT`, it **must** add an instance of the structure, with the desired feature members set to `VK_TRUE`, to the `pNext` chain of `VkDeviceCreateInfo` when creating the `VkDevice`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceColorWriteEnableFeaturesEXT-sType-sType **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COLOR_WRITE_ENABLE_FEATURES_EXT`

33.1. Feature Requirements

All Vulkan graphics implementations **must** support the following features:

- If Vulkan 1.0 is supported, `robustBufferAccess` **must** be supported
- If Vulkan 1.1 is supported, the following features **must** be supported:
 - `multiview` if Vulkan SC 1.0 is not advertised (see [SCID-8])
 - `storageBuffer16BitAccess` if `uniformAndStorageBuffer16BitAccess` is supported
 - `shaderDrawParameters` if `VK_KHR_shader_draw_parameters` is supported

- If Vulkan 1.2 is supported, the following features **must** be supported:
 - `subgroupBroadcastDynamicId`
 - `imagelessFramebuffer`
 - `uniformBufferStandardLayout`
 - `shaderSubgroupExtendedTypes`
 - `separateDepthStencilLayouts`
 - `hostQueryReset`
 - `timelineSemaphore` if Vulkan SC 1.0 is not advertised (see [\[SCID-8\]](#))
 - `samplerMirrorClampToEdge` if `VK_KHR_sampler_mirror_clamp_to_edge` is supported
 - `drawIndirectCount` if `VK_KHR_draw_indirect_count` is supported
 - `storageBuffer8BitAccess` if `uniformAndStorageBuffer8BitAccess` is supported
 - `shaderInt64` if `shaderSharedInt64Atomics` or `shaderBufferInt64Atomics` are supported
 - `descriptorIndexing` if `VK_EXT_descriptor_indexing` is supported
 - `shaderSampledImageArrayDynamicIndexing` if `descriptorIndexing` is supported
 - `shaderStorageBufferArrayDynamicIndexing` if `descriptorIndexing` is supported
 - `shaderUniformTexelBufferArrayDynamicIndexing` if `descriptorIndexing` is supported
 - `shaderStorageTexelBufferArrayDynamicIndexing` if `descriptorIndexing` is supported
 - `shaderSampledImageArrayNonUniformIndexing` if `descriptorIndexing` is supported
 - `shaderStorageBufferArrayNonUniformIndexing` if `descriptorIndexing` is supported
 - `shaderUniformTexelBufferArrayNonUniformIndexing` if `descriptorIndexing` is supported
 - `descriptorBindingSampledImageUpdateAfterBind` if `descriptorIndexing` is supported
 - `descriptorBindingStorageImageUpdateAfterBind` if `descriptorIndexing` is supported
 - `descriptorBindingStorageBufferUpdateAfterBind` if `descriptorIndexing` is supported
 - `descriptorBindingUniformTexelBufferUpdateAfterBind` if `descriptorIndexing` is supported
 - `descriptorBindingStorageTexelBufferUpdateAfterBind` if `descriptorIndexing` is supported
 - `descriptorBindingUpdateUnusedWhilePending` if `descriptorIndexing` is supported
 - `descriptorBindingPartiallyBound` if `descriptorIndexing` is supported
 - `runtimeDescriptorArray` if `descriptorIndexing` is supported
 - `shaderOutputViewportIndex` if `VK_EXT_shader_viewport_index_layer` is supported
 - `shaderOutputLayer` if `VK_EXT_shader_viewport_index_layer` is supported
 - `samplerFilterMinmax` if `VK_EXT_sampler_filter_minmax` is supported
- If Vulkan SC 1.0 is supported, `vulkanMemoryModel` **must** be supported if Vulkan 1.2 is supported
- If `VK_KHR_performance_query` is supported, `performanceCounterQueryPools` **must** be supported
- If `VK_KHR_shader_clock` is supported, `shaderSubgroupClock` **must** be supported
- If `VK_KHR_global_priority` is supported, `globalPriorityQuery` **must** be supported

- If `VK_KHR_shader_terminate_invocation` is supported, `shaderTerminateInvocation` **must** be supported
- If `VK_KHR_fragment_shading_rate` is supported, `pipelineFragmentShadingRate` **must** be supported
- If `VK_KHR_synchronization2` is supported, `synchronization2` **must** be supported
- If `VK_KHR_vertex_attribute_divisor` is supported, `vertexAttributeInstanceRateDivisor` **must** be supported
- If `VK_KHR_index_type_uint8` is supported, `indexTypeUint8` **must** be supported
- If `VK_KHR_line_rasterization` is supported, at least one of `rectangularLines`, `bresenhamLines`, `smoothLines`, `stippledRectangularLines`, `stippledBresenhamLines`, or `stippledSmoothLines` **must** be supported
- If `VK_EXT_texture_compression_astc_hdr` is supported, `textureCompressionASTC_HDR` **must** be supported
- If `VK_EXT_depth_clip_enable` is supported, `depthClipEnable` **must** be supported
- If `VK_EXT_subgroup_size_control` is supported, the following features **must** be supported:
 - `subgroupSizeControl`
 - `computeFullSubgroups`
- If `VK_EXT_shader_image_atomic_int64` is supported, the following features **must** be supported:
 - `shaderInt64`
 - `shaderImageInt64Atomics`
- If `VK_EXT_fragment_shader_interlock` is supported, at least one of `fragmentShaderSampleInterlock`, `fragmentShaderPixelInterlock`, or `fragmentShaderShadingRateInterlock` **must** be supported
- If `VK_EXT_ycbcr_image_arrays` is supported, `ycbcrImageArrays` **must** be supported
- If `VK_EXT_line_rasterization` is supported, at least one of `rectangularLines`, `bresenhamLines`, `smoothLines`, `stippledRectangularLines`, `stippledBresenhamLines`, or `stippledSmoothLines` **must** be supported
- If `VK_EXT_shader_atomic_float` is supported, the following features **must** be supported:
 - at least one of `shaderBufferFloat32Atomics`, `shaderBufferFloat32AtomicAdd`, `shaderBufferFloat64Atomics`, `shaderBufferFloat64AtomicAdd`, `shaderSharedFloat32Atomics`, `shaderSharedFloat32AtomicAdd`, `shaderSharedFloat64Atomics`, `shaderSharedFloat64AtomicAdd`, `shaderImageFloat32Atomics`, or `shaderImageFloat32AtomicAdd`
 - `shaderImageFloat32Atomics` if `sparseImageFloat32Atomics` is supported
 - `shaderImageFloat32AtomicAdd` if `sparseImageFloat32AtomicAdd` is supported
- If `VK_EXT_index_type_uint8` is supported, `indexTypeUint8` **must** be supported
- If `VK_EXT_extended_dynamic_state` is supported, `extendedDynamicState` **must** be supported
- If `VK_EXT_shader_demote_to_helper_invocation` is supported, `shaderDemoteToHelperInvocation` **must** be supported
- If `VK_EXT_texel_buffer_alignment` is supported, `texelBufferAlignment` **must** be supported
- If `VK_EXT_robustness2` is supported, at least one of `robustBufferAccess2`, `robustImageAccess2`, or

`nullDescriptor` **must** be supported

- If `VK_EXT_custom_border_color` is supported, `customBorderColors` **must** be supported
- If `VK_EXT_ycbcr_2plane_444_formats` is supported, `ycbcr2plane444Formats` **must** be supported
- If `VK_EXT_image_robustness` is supported, `robustImageAccess` **must** be supported
- If `VK_EXT_4444_formats` is supported, `formatA4R4G4B4` **must** be supported
- If `VK_EXT_vertex_input_dynamic_state` is supported, `vertexInputDynamicState` **must** be supported
- If `VK_NV_external_sci_sync` is supported, the following features **must** be supported:
 - at least one of `sciSyncFence` or `sciSyncSemaphore`
 - at least one of `sciSyncImport` or `sciSyncExport`
- If `VK_NV_external_memory_sci_buf` is supported, at least one of `sciBufImport` or `sciBufExport` **must** be supported
- If `VK_EXT_extended_dynamic_state2` is supported, `extendedDynamicState2` **must** be supported
- If `VK_EXT_color_write_enable` is supported, `colorWriteEnable` **must** be supported
- If `VK_NV_external_sci_sync2` is supported, the following features **must** be supported:
 - at least one of `VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncFence` or `sciSyncSemaphore2`
 - at least one of `VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncImport` or `VkPhysicalDeviceExternalSciSync2FeaturesNV::sciSyncExport`
- If `VK_QNX_external_memory_screen_buffer` is supported, `screenBufferImport` **must** be supported

All other features defined in the Specification are **optional**.

Chapter 34. Limits

Limits are implementation-dependent minimums, maximums, and other device characteristics that an application **may** need to be aware of.

NOTE

Limits are reported via the basic [VkPhysicalDeviceLimits](#) structure as well as the extensible structure [VkPhysicalDeviceProperties2](#), which was added in [VK_KHR_get_physical_device_properties2](#) and included in Vulkan 1.1. When limits are added in future Vulkan versions or extensions, each extension **should** introduce one new limit structure, if needed. This structure **can** be added to the [pNext](#) chain of the [VkPhysicalDeviceProperties2](#) structure.

The [VkPhysicalDeviceLimits](#) structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkPhysicalDeviceLimits {
    uint32_t          maxImageDimension1D;
    uint32_t          maxImageDimension2D;
    uint32_t          maxImageDimension3D;
    uint32_t          maxImageDimensionCube;
    uint32_t          maxImageArrayLayers;
    uint32_t          maxTexelBufferElements;
    uint32_t          maxUniformBufferRange;
    uint32_t          maxStorageBufferRange;
    uint32_t          maxPushConstantsSize;
    uint32_t          maxMemoryAllocationCount;
    uint32_t          maxSamplerAllocationCount;
    VkDeviceSize      bufferImageGranularity;
    VkDeviceSize      sparseAddressSpaceSize;
    uint32_t          maxBoundDescriptorSets;
    uint32_t          maxPerStageDescriptorSamplers;
    uint32_t          maxPerStageDescriptorUniformBuffers;
    uint32_t          maxPerStageDescriptorStorageBuffers;
    uint32_t          maxPerStageDescriptorSampledImages;
    uint32_t          maxPerStageDescriptorStorageImages;
    uint32_t          maxPerStageDescriptorInputAttachments;
    uint32_t          maxPerStageResources;
    uint32_t          maxDescriptorSetSamplers;
    uint32_t          maxDescriptorSetUniformBuffers;
    uint32_t          maxDescriptorSetUniformBuffersDynamic;
    uint32_t          maxDescriptorSetStorageBuffers;
    uint32_t          maxDescriptorSetStorageBuffersDynamic;
    uint32_t          maxDescriptorSetSampledImages;
    uint32_t          maxDescriptorSetStorageImages;
    uint32_t          maxDescriptorSetInputAttachments;
    uint32_t          maxVertexInputAttributes;
    uint32_t          maxVertexInputBindings;
    uint32_t          maxVertexInputAttributeOffset;
    uint32_t          maxVertexInputBindingStride;
```

uint32_t	maxVertexOutputComponents;
uint32_t	maxTessellationGenerationLevel;
uint32_t	maxTessellationPatchSize;
uint32_t	maxTessellationControlPerVertexInputComponents;
uint32_t	maxTessellationControlPerVertexOutputComponents;
uint32_t	maxTessellationControlPerPatchOutputComponents;
uint32_t	maxTessellationControlTotalOutputComponents;
uint32_t	maxTessellationEvaluationInputComponents;
uint32_t	maxTessellationEvaluationOutputComponents;
uint32_t	maxGeometryShaderInvocations;
uint32_t	maxGeometryInputComponents;
uint32_t	maxGeometryOutputComponents;
uint32_t	maxGeometryOutputVertices;
uint32_t	maxGeometryTotalOutputComponents;
uint32_t	maxFragmentInputComponents;
uint32_t	maxFragmentOutputAttachments;
uint32_t	maxFragmentDualSrcAttachments;
uint32_t	maxFragmentCombinedOutputResources;
uint32_t	maxComputeSharedMemorySize;
uint32_t	maxComputeWorkGroupCount[3];
uint32_t	maxComputeWorkGroupInvocations;
uint32_t	maxComputeWorkGroupSize[3];
uint32_t	subPixelPrecisionBits;
uint32_t	subTexelPrecisionBits;
uint32_t	mipmapPrecisionBits;
uint32_t	maxDrawIndexedIndexValue;
uint32_t	maxDrawIndirectCount;
float	maxSamplerLodBias;
float	maxSamplerAnisotropy;
uint32_t	maxViewports;
uint32_t	maxViewportDimensions[2];
float	viewportBoundsRange[2];
uint32_t	viewportSubPixelBits;
size_t	minMemoryMapAlignment;
VkDeviceSize	minTexelBufferOffsetAlignment;
VkDeviceSize	minUniformBufferOffsetAlignment;
VkDeviceSize	minStorageBufferOffsetAlignment;
int32_t	minTexelOffset;
uint32_t	maxTexelOffset;
int32_t	minTexelGatherOffset;
uint32_t	maxTexelGatherOffset;
float	minInterpolationOffset;
float	maxInterpolationOffset;
uint32_t	subPixelInterpolationOffsetBits;
uint32_t	maxFramebufferWidth;
uint32_t	maxFramebufferHeight;
uint32_t	maxFramebufferLayers;
VkSampleCountFlags	framebufferColorSampleCounts;
VkSampleCountFlags	framebufferDepthSampleCounts;
VkSampleCountFlags	framebufferStencilSampleCounts;
VkSampleCountFlags	framebufferNoAttachmentsSampleCounts;

```

uint32_t      maxColorAttachments;
VkSampleCountFlags sampledImageColorSampleCounts;
VkSampleCountFlags sampledImageIntegerSampleCounts;
VkSampleCountFlags sampledImageDepthSampleCounts;
VkSampleCountFlags sampledImageStencilSampleCounts;
VkSampleCountFlags storageImageSampleCounts;
uint32_t      maxSampleMaskWords;
VkBool32      timestampComputeAndGraphics;
float         timestampPeriod;
uint32_t      maxClipDistances;
uint32_t      maxCullDistances;
uint32_t      maxCombinedClipAndCullDistances;
uint32_t      discreteQueuePriorities;
float         pointSizeRange[2];
float         lineWidthRange[2];
float         pointSizeGranularity;
float         lineWidthGranularity;
VkBool32      strictLines;
VkBool32      standardSampleLocations;
VkDeviceSize  optimalBufferCopyOffsetAlignment;
VkDeviceSize  optimalBufferCopyRowPitchAlignment;
VkDeviceSize  nonCoherentAtomSize;
} VkPhysicalDeviceLimits;

```

The `VkPhysicalDeviceLimits` are properties of the physical device. These are available in the `limits` member of the `VkPhysicalDeviceProperties` structure which is returned from `vkGetPhysicalDeviceProperties`.

- `maxImageDimension1D` is the largest dimension (`width`) that is guaranteed to be supported for all images created with an `imageType` of `VK_IMAGE_TYPE_1D`. Some combinations of image parameters (format, usage, etc.) **may** allow support for larger dimensions, which **can** be queried using `vkGetPhysicalDeviceImageFormatProperties`.
- `maxImageDimension2D` is the largest dimension (`width` or `height`) that is guaranteed to be supported for all images created with an `imageType` of `VK_IMAGE_TYPE_2D` and without `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT` set in `flags`. Some combinations of image parameters (format, usage, etc.) **may** allow support for larger dimensions, which **can** be queried using `vkGetPhysicalDeviceImageFormatProperties`.
- `maxImageDimension3D` is the largest dimension (`width`, `height`, or `depth`) that is guaranteed to be supported for all images created with an `imageType` of `VK_IMAGE_TYPE_3D`. Some combinations of image parameters (format, usage, etc.) **may** allow support for larger dimensions, which **can** be queried using `vkGetPhysicalDeviceImageFormatProperties`.
- `maxImageDimensionCube` is the largest dimension (`width` or `height`) that is guaranteed to be supported for all images created with an `imageType` of `VK_IMAGE_TYPE_2D` and with `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT` set in `flags`. Some combinations of image parameters (format, usage, etc.) **may** allow support for larger dimensions, which **can** be queried using `vkGetPhysicalDeviceImageFormatProperties`.
- `maxImageArrayLayers` is the maximum number of layers (`arrayLayers`) for an image.

- `maxTexelBufferElements` is the maximum number of addressable texels for a buffer view created on a buffer which was created with the `VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT` or `VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT` set in the `usage` member of the `VkBufferCreateInfo` structure.
- `maxUniformBufferRange` is the maximum value that **can** be specified in the `range` member of a `VkDescriptorBufferInfo` structure passed to `vkUpdateDescriptorSets` for descriptors of type `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`.
- `maxStorageBufferRange` is the maximum value that **can** be specified in the `range` member of a `VkDescriptorBufferInfo` structure passed to `vkUpdateDescriptorSets` for descriptors of type `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`.
- `maxPushConstantsSize` is the maximum size, in bytes, of the pool of push constant memory. For each of the push constant ranges indicated by the `pPushConstantRanges` member of the `VkPipelineLayoutCreateInfo` structure, $(\text{offset} + \text{size})$ **must** be less than or equal to this limit.
- `maxMemoryAllocationCount` is the maximum number of device memory allocations, as created by `vkAllocateMemory`, which **can** simultaneously exist.
- `maxSamplerAllocationCount` is the maximum number of sampler objects, as created by `vkCreateSampler`, which **can** simultaneously exist on a device.
- `bufferImageGranularity` is the granularity, in bytes, at which buffer or linear image resources, and optimal image resources **can** be bound to adjacent offsets in the same `VkDeviceMemory` object without aliasing. See [Buffer-Image Granularity](#) for more details.
- `sparseAddressSpaceSize` is the total amount of address space available, in bytes, for sparse memory resources. This is an upper bound on the sum of the sizes of all sparse resources, regardless of whether any memory is bound to them.
- `maxBoundDescriptorSets` is the maximum number of descriptor sets that **can** be simultaneously used by a pipeline. All `DescriptorSet` decorations in shader modules **must** have a value less than `maxBoundDescriptorSets`. See [Descriptor Sets](#).
- `maxPerStageDescriptorSamplers` is the maximum number of samplers that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_SAMPLER` or `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. A descriptor is accessible to a shader stage when the `stageFlags` member of the `VkDescriptorSetLayoutBinding` structure has the bit for that shader stage set. See [Sampler](#) and [Combined Image Sampler](#).
- `maxPerStageDescriptorUniformBuffers` is the maximum number of uniform buffers that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. A descriptor is accessible to a shader stage when the `stageFlags` member of the `VkDescriptorSetLayoutBinding` structure has the bit for that shader stage set. See [Uniform Buffer](#) and [Dynamic Uniform Buffer](#).
- `maxPerStageDescriptorStorageBuffers` is the maximum number of storage buffers that **can** be

accessible to a single shader stage in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. A descriptor is accessible to a pipeline shader stage when the `stageFlags` member of the `VkDescriptorSetLayoutBinding` structure has the bit for that shader stage set. See [Storage Buffer](#) and [Dynamic Storage Buffer](#).

- `maxPerStageDescriptorSampledImages` is the maximum number of sampled images that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, or `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. A descriptor is accessible to a pipeline shader stage when the `stageFlags` member of the `VkDescriptorSetLayoutBinding` structure has the bit for that shader stage set. See [Combined Image Sampler](#), [Sampled Image](#), and [Uniform Texel Buffer](#).
- `maxPerStageDescriptorStorageImages` is the maximum number of storage images that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. A descriptor is accessible to a pipeline shader stage when the `stageFlags` member of the `VkDescriptorSetLayoutBinding` structure has the bit for that shader stage set. See [Storage Image](#), and [Storage Texel Buffer](#).
- `maxPerStageDescriptorInputAttachments` is the maximum number of input attachments that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. A descriptor is accessible to a pipeline shader stage when the `stageFlags` member of the `VkDescriptorSetLayoutBinding` structure has the bit for that shader stage set. These are only supported for the fragment stage. See [Input Attachment](#).
- `maxPerStageResources` is the maximum number of resources that **can** be accessible to a single shader stage in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER`, `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER`, `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER`, `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER`, `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`, `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC`, or `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. For the fragment shader stage the framebuffer color attachments also count against this limit.
- `maxDescriptorSetSamplers` is the maximum number of samplers that **can** be included in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_SAMPLER` or `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. See [Sampler](#) and [Combined Image Sampler](#).

- `maxDescriptorSetUniformBuffers` is the maximum number of uniform buffers that **can** be included in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. See [Uniform Buffer](#) and [Dynamic Uniform Buffer](#).
- `maxDescriptorSetUniformBuffersDynamic` is the maximum number of dynamic uniform buffers that **can** be included in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. See [Dynamic Uniform Buffer](#).
- `maxDescriptorSetStorageBuffers` is the maximum number of storage buffers that **can** be included in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. See [Storage Buffer](#) and [Dynamic Storage Buffer](#).
- `maxDescriptorSetStorageBuffersDynamic` is the maximum number of dynamic storage buffers that **can** be included in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. See [Dynamic Storage Buffer](#).
- `maxDescriptorSetSampledImages` is the maximum number of sampled images that **can** be included in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER`, `VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE`, or `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. See [Combined Image Sampler](#), [Sampled Image](#), and [Uniform Texel Buffer](#).
- `maxDescriptorSetStorageImages` is the maximum number of storage images that **can** be included in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_STORAGE_IMAGE`, or `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. See [Storage Image](#), and [Storage Texel Buffer](#).
- `maxDescriptorSetInputAttachments` is the maximum number of input attachments that **can** be included in a pipeline layout. Descriptors with a type of `VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT` count against this limit. Only descriptors in descriptor set layouts created without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set count against this limit. See [Input Attachment](#).
- `maxVertexInputAttributes` is the maximum number of vertex input attributes that **can** be specified for a graphics pipeline. These are described in the array of `VkVertexInputAttributeDescription` structures that are provided at graphics pipeline creation time via the `pVertexAttributeDescriptions` member of the `VkPipelineVertexInputStateCreateInfo` structure. See [Vertex Attributes](#) and [Vertex Input Description](#).

- `maxVertexInputBindings` is the maximum number of vertex buffers that **can** be specified for providing vertex attributes to a graphics pipeline. These are described in the array of `VkVertexInputBindingDescription` structures that are provided at graphics pipeline creation time via the `pVertexBindingDescriptions` member of the `VkPipelineVertexInputStateCreateInfo` structure. The `binding` member of `VkVertexInputBindingDescription` **must** be less than this limit. See [Vertex Input Description](#).
- `maxVertexInputAttributeOffset` is the maximum vertex input attribute offset that **can** be added to the vertex input binding stride. The `offset` member of the `VkVertexInputAttributeDescription` structure **must** be less than or equal to this limit. See [Vertex Input Description](#).
- `maxVertexInputBindingStride` is the maximum vertex input binding stride that **can** be specified in a vertex input binding. The `stride` member of the `VkVertexInputBindingDescription` structure **must** be less than or equal to this limit. See [Vertex Input Description](#).
- `maxVertexOutputComponents` is the maximum number of components of output variables which **can** be output by a vertex shader. See [Vertex Shaders](#).
- `maxTessellationGenerationLevel` is the maximum tessellation generation level supported by the fixed-function tessellation primitive generator. See [Tessellation](#).
- `maxTessellationPatchSize` is the maximum patch size, in vertices, of patches that **can** be processed by the tessellation control shader and tessellation primitive generator. The `patchControlPoints` member of the `VkPipelineTessellationStateCreateInfo` structure specified at pipeline creation time and the value provided in the `OutputVertices` execution mode of shader modules **must** be less than or equal to this limit. See [Tessellation](#).
- `maxTessellationControlPerVertexInputComponents` is the maximum number of components of input variables which **can** be provided as per-vertex inputs to the tessellation control shader stage.
- `maxTessellationControlPerVertexOutputComponents` is the maximum number of components of per-vertex output variables which **can** be output from the tessellation control shader stage.
- `maxTessellationControlPerPatchOutputComponents` is the maximum number of components of per-patch output variables which **can** be output from the tessellation control shader stage.
- `maxTessellationControlTotalOutputComponents` is the maximum total number of components of per-vertex and per-patch output variables which **can** be output from the tessellation control shader stage.
- `maxTessellationEvaluationInputComponents` is the maximum number of components of input variables which **can** be provided as per-vertex inputs to the tessellation evaluation shader stage.
- `maxTessellationEvaluationOutputComponents` is the maximum number of components of per-vertex output variables which **can** be output from the tessellation evaluation shader stage.
- `maxGeometryShaderInvocations` is the maximum invocation count supported for instanced geometry shaders. The value provided in the `Invocations` execution mode of shader modules **must** be less than or equal to this limit. See [Geometry Shading](#).
- `maxGeometryInputComponents` is the maximum number of components of input variables which **can** be provided as inputs to the geometry shader stage.
- `maxGeometryOutputComponents` is the maximum number of components of output variables which

can be output from the geometry shader stage.

- **maxGeometryOutputVertices** is the maximum number of vertices which **can** be emitted by any geometry shader.
- **maxGeometryTotalOutputComponents** is the maximum total number of components of output variables, across all emitted vertices, which **can** be output from the geometry shader stage.
- **maxFragmentInputComponents** is the maximum number of components of input variables which **can** be provided as inputs to the fragment shader stage.
- **maxFragmentOutputAttachments** is the maximum number of output attachments which **can** be written to by the fragment shader stage.
- **maxFragmentDualSrcAttachments** is the maximum number of output attachments which **can** be written to by the fragment shader stage when blending is enabled and one of the dual source blend modes is in use. See [Dual-Source Blending](#) and **dualSrcBlend**.
- **maxFragmentCombinedOutputResources** is the total number of storage buffers, storage images, and output **Location** decorated color attachments (described in [Fragment Output Interface](#)) which **can** be used in the fragment shader stage.
- **maxComputeSharedMemorySize** is the maximum total storage size, in bytes, available for variables declared with the **Workgroup** storage class in shader modules (or with the **shared** storage qualifier in GLSL) in the compute shader stage.
- **maxComputeWorkGroupCount**[3] is the maximum number of local workgroups that **can** be dispatched by a single dispatching command. These three values represent the maximum number of local workgroups for the X, Y, and Z dimensions, respectively. The workgroup count parameters to the dispatching commands **must** be less than or equal to the corresponding limit. See [Dispatching Commands](#).
- **maxComputeWorkGroupInvocations** is the maximum total number of compute shader invocations in a single local workgroup. The product of the X, Y, and Z sizes, as specified by the **LocalSize** execution mode in shader modules or by the object decorated by the **WorkgroupSize** decoration, **must** be less than or equal to this limit.
- **maxComputeWorkGroupSize**[3] is the maximum size of a local compute workgroup, per dimension. These three values represent the maximum local workgroup size in the X, Y, and Z dimensions, respectively. The **x**, **y**, and **z** sizes, as specified by the **LocalSize** execution mode or by the object decorated by the **WorkgroupSize** decoration in shader modules, **must** be less than or equal to the corresponding limit.
- **subPixelPrecisionBits** is the number of bits of subpixel precision in framebuffer coordinates x_f and y_f . See [Rasterization](#).
- **subTexelPrecisionBits** is the number of bits of precision in the division along an axis of an image used for minification and magnification filters. $2^{\text{subTexelPrecisionBits}}$ is the actual number of divisions along each axis of the image represented. Sub-texel values calculated during image sampling will snap to these locations when generating the filtered results.
- **mipmapPrecisionBits** is the number of bits of division that the LOD calculation for mipmap fetching get snapped to when determining the contribution from each mip level to the mip filtered results. $2^{\text{mipmapPrecisionBits}}$ is the actual number of divisions.
- **maxDrawIndexedIndexValue** is the maximum index value that **can** be used for indexed draw calls

when using 32-bit indices. This excludes the primitive restart index value of 0xFFFFFFFF. See [fullDrawIndexUint32](#).

- **maxDrawIndirectCount** is the maximum draw count that is supported for indirect drawing calls. See [multiDrawIndirect](#).
- **maxSamplerLodBias** is the maximum absolute sampler LOD bias. The sum of the **mipLodBias** member of the [VkSamplerCreateInfo](#) structure and the **Bias** operand of image sampling operations in shader modules (or 0 if no **Bias** operand is provided to an image sampling operation) are clamped to the range $[-\text{maxSamplerLodBias}, +\text{maxSamplerLodBias}]$. See [\[samplers-mipLodBias\]](#).
- **maxSamplerAnisotropy** is the maximum degree of sampler anisotropy. The maximum degree of anisotropic filtering used for an image sampling operation is the minimum of the **maxAnisotropy** member of the [VkSamplerCreateInfo](#) structure and this limit. See [\[samplers-maxAnisotropy\]](#).
- **maxViewports** is the maximum number of active viewports. The **viewportCount** member of the [VkPipelineViewportStateCreateInfo](#) structure that is provided at pipeline creation **must** be less than or equal to this limit.
- **maxViewportDimensions[2]** are the maximum viewport dimensions in the X (width) and Y (height) dimensions, respectively. The maximum viewport dimensions **must** be greater than or equal to the largest image which **can** be created and used as a framebuffer attachment. See [Controlling the Viewport](#).
- **viewportBoundsRange[2]** is the [minimum, maximum] range that the corners of a viewport **must** be contained in. This range **must** be at least $[-2 \times \text{size}, 2 \times \text{size} - 1]$, where **size** = $\max(\text{maxViewportDimensions}[0], \text{maxViewportDimensions}[1])$. See [Controlling the Viewport](#).

NOTE

The intent of the **viewportBoundsRange** limit is to allow a maximum sized viewport to be arbitrarily shifted relative to the output target as long as at least some portion intersects. This would give a bounds limit of $[-\text{size} + 1, 2 \times \text{size} - 1]$ which would allow all possible non-empty-set intersections of the output target and the viewport. Since these numbers are typically powers of two, picking the signed number range using the smallest possible number of bits ends up with the specified range.

- **viewportSubPixelBits** is the number of bits of subpixel precision for viewport bounds. The subpixel precision that floating-point viewport bounds are interpreted at is given by this limit.
- **minMemoryMapAlignment** is the minimum **required** alignment, in bytes, of host visible memory allocations within the host address space. When mapping a memory allocation with [vkMapMemory](#), subtracting **offset** bytes from the returned pointer will always produce an integer multiple of this limit. See [Host Access to Device Memory Objects](#). The value **must** be a power of two.
- **minTexelBufferOffsetAlignment** is the minimum **required** alignment, in bytes, for the **offset** member of the [VkBufferViewCreateInfo](#) structure for texel buffers. The value **must** be a power of two. If the **texelBufferAlignment** feature is enabled, this limit is equivalent to the maximum of the **uniformTexelBufferOffsetAlignmentBytes** and **storageTexelBufferOffsetAlignmentBytes** members of [VkPhysicalDeviceTexelBufferAlignmentProperties](#), but smaller alignment is **optionally** allowed by **storageTexelBufferOffsetSingleTexelAlignment** and

`uniformTexelBufferOffsetSingleTexelAlignment`. If the `texelBufferAlignment` feature is not enabled, `VkBufferViewCreateInfo::offset` **must** be a multiple of this value.

- `minUniformBufferOffsetAlignment` is the minimum **required** alignment, in bytes, for the `offset` member of the `VkDescriptorBufferInfo` structure for uniform buffers. When a descriptor of type `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER` or `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC` is updated, the `offset` **must** be an integer multiple of this limit. Similarly, dynamic offsets for uniform buffers **must** be multiples of this limit. The value **must** be a power of two.
- `minStorageBufferOffsetAlignment` is the minimum **required** alignment, in bytes, for the `offset` member of the `VkDescriptorBufferInfo` structure for storage buffers. When a descriptor of type `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER` or `VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC` is updated, the `offset` **must** be an integer multiple of this limit. Similarly, dynamic offsets for storage buffers **must** be multiples of this limit. The value **must** be a power of two.
- `minTexelOffset` is the minimum offset value for the `ConstOffset` image operand of any of the `OpImageSample*` or `OpImageFetch*` image instructions.
- `maxTexelOffset` is the maximum offset value for the `ConstOffset` image operand of any of the `OpImageSample*` or `OpImageFetch*` image instructions.
- `minTexelGatherOffset` is the minimum offset value for the `Offset`, `ConstOffset`, or `ConstOffsets` image operands of any of the `OpImage*Gather` image instructions.
- `maxTexelGatherOffset` is the maximum offset value for the `Offset`, `ConstOffset`, or `ConstOffsets` image operands of any of the `OpImage*Gather` image instructions.
- `minInterpolationOffset` is the base minimum (inclusive) negative offset value for the `Offset` operand of the `InterpolateAtOffset` extended instruction.
- `maxInterpolationOffset` is the base maximum (inclusive) positive offset value for the `Offset` operand of the `InterpolateAtOffset` extended instruction.
- `subPixelInterpolationOffsetBits` is the number of fractional bits that the `x` and `y` offsets to the `InterpolateAtOffset` extended instruction **may** be rounded to as fixed-point values.
- `maxFramebufferWidth` is the maximum width for a framebuffer. The `width` member of the `VkFramebufferCreateInfo` structure **must** be less than or equal to this limit.
- `maxFramebufferHeight` is the maximum height for a framebuffer. The `height` member of the `VkFramebufferCreateInfo` structure **must** be less than or equal to this limit.
- `maxFramebufferLayers` is the maximum layer count for a layered framebuffer. The `layers` member of the `VkFramebufferCreateInfo` structure **must** be less than or equal to this limit.
- `framebufferColorSampleCounts` is a bitmask¹ of `VkSampleCountFlagBits` indicating the color sample counts that are supported for all framebuffer color attachments with floating- or fixed-point formats. For color attachments with integer formats, see `framebufferIntegerColorSampleCounts`.
- `framebufferDepthSampleCounts` is a bitmask¹ of `VkSampleCountFlagBits` indicating the supported depth sample counts for all framebuffer depth/stencil attachments, when the format includes a depth component.
- `framebufferStencilSampleCounts` is a bitmask¹ of `VkSampleCountFlagBits` indicating the supported stencil sample counts for all framebuffer depth/stencil attachments, when the format includes a stencil component.

- `framebufferNoAttachmentsSampleCounts` is a bitmask¹ of `VkSampleCountFlagBits` indicating the supported sample counts for a [subpass which uses no attachments](#).
- `maxColorAttachments` is the maximum number of color attachments that **can** be used by a subpass in a render pass. The `colorAttachmentCount` member of the `VkSubpassDescription` or `VkSubpassDescription2` structure **must** be less than or equal to this limit.
- `sampledImageColorSampleCounts` is a bitmask¹ of `VkSampleCountFlagBits` indicating the sample counts supported for all 2D images created with `VK_IMAGE_TILING_OPTIMAL`, `usage` containing `VK_IMAGE_USAGE_SAMPLED_BIT`, and a non-integer color format.
- `sampledImageIntegerSampleCounts` is a bitmask¹ of `VkSampleCountFlagBits` indicating the sample counts supported for all 2D images created with `VK_IMAGE_TILING_OPTIMAL`, `usage` containing `VK_IMAGE_USAGE_SAMPLED_BIT`, and an integer color format.
- `sampledImageDepthSampleCounts` is a bitmask¹ of `VkSampleCountFlagBits` indicating the sample counts supported for all 2D images created with `VK_IMAGE_TILING_OPTIMAL`, `usage` containing `VK_IMAGE_USAGE_SAMPLED_BIT`, and a depth format.
- `sampledImageStencilSampleCounts` is a bitmask¹ of `VkSampleCountFlagBits` indicating the sample counts supported for all 2D images created with `VK_IMAGE_TILING_OPTIMAL`, `usage` containing `VK_IMAGE_USAGE_SAMPLED_BIT`, and a stencil format.
- `storageImageSampleCounts` is a bitmask¹ of `VkSampleCountFlagBits` indicating the sample counts supported for all 2D images created with `VK_IMAGE_TILING_OPTIMAL`, and `usage` containing `VK_IMAGE_USAGE_STORAGE_BIT`.
- `maxSampleMaskWords` is the maximum number of array elements of a variable decorated with the `SampleMask` built-in decoration.
- `timestampComputeAndGraphics` specifies support for timestamps on all graphics and compute queues. If this limit is `VK_TRUE`, all queues that advertise the `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT` in the `VkQueueFamilyProperties::queueFlags` support `VkQueueFamilyProperties::timestampValidBits` of at least 36. See [Timestamp Queries](#).
- `timestampPeriod` is the number of nanoseconds **required** for a timestamp query to be incremented by 1. See [Timestamp Queries](#).
- `maxClipDistances` is the maximum number of clip distances that **can** be used in a single shader stage. The size of any array declared with the `ClipDistance` built-in decoration in a shader module **must** be less than or equal to this limit.
- `maxCullDistances` is the maximum number of cull distances that **can** be used in a single shader stage. The size of any array declared with the `CullDistance` built-in decoration in a shader module **must** be less than or equal to this limit.
- `maxCombinedClipAndCullDistances` is the maximum combined number of clip and cull distances that **can** be used in a single shader stage. The sum of the sizes of all arrays declared with the `ClipDistance` and `CullDistance` built-in decoration used by a single shader stage in a shader module **must** be less than or equal to this limit.
- `discreteQueuePriorities` is the number of discrete priorities that **can** be assigned to a queue based on the value of each member of `VkDeviceQueueCreateInfo::pQueuePriorities`. This **must** be at least 2, and levels **must** be spread evenly over the range, with at least one level at 1.0, and another at 0.0. See [Queue Priority](#).

- `pointSizeRange[2]` is the range `[minimum,maximum]` of supported sizes for points. Values written to variables decorated with the `PointSize` built-in decoration are clamped to this range.
- `lineWidthRange[2]` is the range `[minimum,maximum]` of supported widths for lines. Values specified by the `lineWidth` member of the `VkPipelineRasterizationStateCreateInfo` or the `lineWidth` parameter to `vkCmdSetLineWidth` are clamped to this range.
- `pointSizeGranularity` is the granularity of supported point sizes. Not all point sizes in the range defined by `pointSizeRange` are supported. This limit specifies the granularity (or increment) between successive supported point sizes.
- `lineWidthGranularity` is the granularity of supported line widths. Not all line widths in the range defined by `lineWidthRange` are supported. This limit specifies the granularity (or increment) between successive supported line widths.
- `strictLines` specifies whether lines are rasterized according to the preferred method of rasterization. If set to `VK_FALSE`, lines **may** be rasterized under a relaxed set of rules. If set to `VK_TRUE`, lines are rasterized as per the strict definition. See [Basic Line Segment Rasterization](#).
- `standardSampleLocations` specifies whether rasterization uses the standard sample locations as documented in [Multisampling](#). If set to `VK_TRUE`, the implementation uses the documented sample locations. If set to `VK_FALSE`, the implementation **may** use different sample locations.
- `optimalBufferCopyOffsetAlignment` is the optimal buffer offset alignment in bytes for `vkCmdCopyBufferToImage2KHR`, `vkCmdCopyBufferToImage`, `vkCmdCopyImageToBuffer2KHR`, and `vkCmdCopyImageToBuffer`. The per texel alignment requirements are enforced, but applications **should** use the optimal alignment for optimal performance and power use. The value **must** be a power of two.
- `optimalBufferCopyRowPitchAlignment` is the optimal buffer row pitch alignment in bytes for `vkCmdCopyBufferToImage2KHR`, `vkCmdCopyBufferToImage`, `vkCmdCopyImageToBuffer2KHR`, and `vkCmdCopyImageToBuffer`. Row pitch is the number of bytes between texels with the same X coordinate in adjacent rows (Y coordinates differ by one). The per texel alignment requirements are enforced, but applications **should** use the optimal alignment for optimal performance and power use. The value **must** be a power of two.
- `nonCoherentAtomSize` is the size and alignment in bytes that bounds concurrent access to [host-mapped device memory](#). The value **must** be a power of two.

1

For all bitmasks of `VkSampleCountFlagBits`, the sample count limits defined above represent the minimum supported sample counts for each image type. Individual images **may** support additional sample counts, which are queried using `vkGetPhysicalDeviceImageFormatProperties` as described in [Supported Sample Counts](#).

Bits which **may** be set in the sample count limits returned by `VkPhysicalDeviceLimits`, as well as in other queries and structures representing image sample counts, are:


```
// Provided by VK_VERSION_1_0
typedef enum VkSampleCountFlagBits {
    VK_SAMPLE_COUNT_1_BIT = 0x00000001,
    VK_SAMPLE_COUNT_2_BIT = 0x00000002,
    VK_SAMPLE_COUNT_4_BIT = 0x00000004,
    VK_SAMPLE_COUNT_8_BIT = 0x00000008,
    VK_SAMPLE_COUNT_16_BIT = 0x00000010,
    VK_SAMPLE_COUNT_32_BIT = 0x00000020,
    VK_SAMPLE_COUNT_64_BIT = 0x00000040,
} VkSampleCountFlagBits;
```

- `VK_SAMPLE_COUNT_1_BIT` specifies an image with one sample per pixel.
- `VK_SAMPLE_COUNT_2_BIT` specifies an image with 2 samples per pixel.
- `VK_SAMPLE_COUNT_4_BIT` specifies an image with 4 samples per pixel.
- `VK_SAMPLE_COUNT_8_BIT` specifies an image with 8 samples per pixel.
- `VK_SAMPLE_COUNT_16_BIT` specifies an image with 16 samples per pixel.
- `VK_SAMPLE_COUNT_32_BIT` specifies an image with 32 samples per pixel.
- `VK_SAMPLE_COUNT_64_BIT` specifies an image with 64 samples per pixel.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkSampleCountFlags;
```

`VkSampleCountFlags` is a bitmask type for setting a mask of zero or more `VkSampleCountFlagBits`.

The `VkPhysicalDeviceMultiviewProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceMultiviewProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t            maxMultiviewViewCount;
    uint32_t            maxMultiviewInstanceIndex;
} VkPhysicalDeviceMultiviewProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxMultiviewViewCount` is one greater than the maximum view index that **can** be used in a subpass.
- `maxMultiviewInstanceIndex` is the maximum valid value of instance index allowed to be generated by a drawing command recorded within a subpass of a multiview render pass instance.

If the `VkPhysicalDeviceMultiviewProperties` structure is included in the `pNext` chain of the

`VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceMultiviewProperties-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PROPERTIES`

The `VkPhysicalDeviceFloatControlsProperties` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceFloatControlsProperties {
    VkStructureType    sType;
    void*              pNext;
    VkShaderFloatControlsIndependence  denormBehaviorIndependence;
    VkShaderFloatControlsIndependence  roundingModeIndependence;
    VkBool32           shaderSignedZeroInfNanPreserveFloat16;
    VkBool32           shaderSignedZeroInfNanPreserveFloat32;
    VkBool32           shaderSignedZeroInfNanPreserveFloat64;
    VkBool32           shaderDenormPreserveFloat16;
    VkBool32           shaderDenormPreserveFloat32;
    VkBool32           shaderDenormPreserveFloat64;
    VkBool32           shaderDenormFlushToZeroFloat16;
    VkBool32           shaderDenormFlushToZeroFloat32;
    VkBool32           shaderDenormFlushToZeroFloat64;
    VkBool32           shaderRoundingModeRTEFloat16;
    VkBool32           shaderRoundingModeRTEFloat32;
    VkBool32           shaderRoundingModeRTEFloat64;
    VkBool32           shaderRoundingModeRTZFloat16;
    VkBool32           shaderRoundingModeRTZFloat32;
    VkBool32           shaderRoundingModeRTZFloat64;
} VkPhysicalDeviceFloatControlsProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `denormBehaviorIndependence` is a `VkShaderFloatControlsIndependence` value indicating whether, and how, denorm behavior can be set independently for different bit widths.
- `roundingModeIndependence` is a `VkShaderFloatControlsIndependence` value indicating whether, and how, rounding modes can be set independently for different bit widths.
- `shaderSignedZeroInfNanPreserveFloat16` is a boolean value indicating whether sign of a zero, Nans and $\pm\infty$ **can** be preserved in 16-bit floating-point computations. It also indicates whether the `SignedZeroInfNanPreserve` execution mode **can** be used for 16-bit floating-point types.
- `shaderSignedZeroInfNanPreserveFloat32` is a boolean value indicating whether sign of a zero, Nans and $\pm\infty$ **can** be preserved in 32-bit floating-point computations. It also indicates whether the `SignedZeroInfNanPreserve` execution mode **can** be used for 32-bit floating-point types.

- `shaderSignedZeroInfNanPreserveFloat64` is a boolean value indicating whether sign of a zero, Nans and $\pm\infty$ **can** be preserved in 64-bit floating-point computations. It also indicates whether the `SignedZeroInfNanPreserve` execution mode **can** be used for 64-bit floating-point types.
- `shaderDenormPreserveFloat16` is a boolean value indicating whether denormals **can** be preserved in 16-bit floating-point computations. It also indicates whether the `DenormPreserve` execution mode **can** be used for 16-bit floating-point types.
- `shaderDenormPreserveFloat32` is a boolean value indicating whether denormals **can** be preserved in 32-bit floating-point computations. It also indicates whether the `DenormPreserve` execution mode **can** be used for 32-bit floating-point types.
- `shaderDenormPreserveFloat64` is a boolean value indicating whether denormals **can** be preserved in 64-bit floating-point computations. It also indicates whether the `DenormPreserve` execution mode **can** be used for 64-bit floating-point types.
- `shaderDenormFlushToZeroFloat16` is a boolean value indicating whether denormals **can** be flushed to zero in 16-bit floating-point computations. It also indicates whether the `DenormFlushToZero` execution mode **can** be used for 16-bit floating-point types.
- `shaderDenormFlushToZeroFloat32` is a boolean value indicating whether denormals **can** be flushed to zero in 32-bit floating-point computations. It also indicates whether the `DenormFlushToZero` execution mode **can** be used for 32-bit floating-point types.
- `shaderDenormFlushToZeroFloat64` is a boolean value indicating whether denormals **can** be flushed to zero in 64-bit floating-point computations. It also indicates whether the `DenormFlushToZero` execution mode **can** be used for 64-bit floating-point types.
- `shaderRoundingModeRTEFloat16` is a boolean value indicating whether an implementation supports the round-to-nearest-even rounding mode for 16-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTE` execution mode **can** be used for 16-bit floating-point types.
- `shaderRoundingModeRTEFloat32` is a boolean value indicating whether an implementation supports the round-to-nearest-even rounding mode for 32-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTE` execution mode **can** be used for 32-bit floating-point types.
- `shaderRoundingModeRTEFloat64` is a boolean value indicating whether an implementation supports the round-to-nearest-even rounding mode for 64-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTE` execution mode **can** be used for 64-bit floating-point types.
- `shaderRoundingModeRTZFloat16` is a boolean value indicating whether an implementation supports the round-towards-zero rounding mode for 16-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTZ` execution mode **can** be used for 16-bit floating-point types.
- `shaderRoundingModeRTZFloat32` is a boolean value indicating whether an implementation supports the round-towards-zero rounding mode for 32-bit floating-point arithmetic and conversion instructions. It also indicates whether the `RoundingModeRTZ` execution mode **can** be used for 32-bit floating-point types.
- `shaderRoundingModeRTZFloat64` is a boolean value indicating whether an implementation supports the round-towards-zero rounding mode for 64-bit floating-point arithmetic and

conversion instructions. It also indicates whether the `RoundingModeRTZ` execution mode **can** be used for 64-bit floating-point types.

If the `VkPhysicalDeviceFloatControlsProperties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceFloatControlsProperties-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FLOAT_CONTROLS_PROPERTIES`

Values which **may** be returned in the `denormBehaviorIndependence` and `roundingModeIndependence` fields of `VkPhysicalDeviceFloatControlsProperties` are:

```
// Provided by VK_VERSION_1_2
typedef enum VkShaderFloatControlsIndependence {
    VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_32_BIT_ONLY = 0,
    VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_ALL = 1,
    VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_NONE = 2,
} VkShaderFloatControlsIndependence;
```

- `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_32_BIT_ONLY` specifies that shader float controls for 32-bit floating-point **can** be set independently; other bit widths **must** be set identically to each other.
- `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_ALL` specifies that shader float controls for all bit widths **can** be set independently.
- `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_NONE` specifies that shader float controls for all bit widths **must** be set identically.

The `VkPhysicalDeviceDiscardRectanglePropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_discard_rectangles
typedef struct VkPhysicalDeviceDiscardRectanglePropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxDiscardRectangles;
} VkPhysicalDeviceDiscardRectanglePropertiesEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxDiscardRectangles` is the maximum number of active discard rectangles that **can** be specified.

If the `VkPhysicalDeviceDiscardRectanglePropertiesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDiscardRectanglePropertiesEXT-sType-sType
sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DISCARD_RECTANGLE_PROPERTIES_EXT

The `VkPhysicalDeviceSampleLocationsPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_sample_locations
typedef struct VkPhysicalDeviceSampleLocationsPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkSampleCountFlags sampleLocationSampleCounts;
    VkExtent2D         maxSampleLocationGridSize;
    float              sampleLocationCoordinateRange[2];
    uint32_t           sampleLocationSubPixelBits;
    VkBool32           variableSampleLocations;
} VkPhysicalDeviceSampleLocationsPropertiesEXT;
```

- sType is a `VkStructureType` value identifying this structure.
- pNext is `NULL` or a pointer to a structure extending this structure.
- sampleLocationSampleCounts is a bitmask of `VkSampleCountFlagBits` indicating the sample counts supporting custom sample locations.
- maxSampleLocationGridSize is the maximum size of the pixel grid in which sample locations **can** vary that is supported for all sample counts in sampleLocationSampleCounts.
- sampleLocationCoordinateRange[2] is the range of supported sample location coordinates.
- sampleLocationSubPixelBits is the number of bits of subpixel precision for sample locations.
- variableSampleLocations specifies whether the sample locations used by all pipelines that will be bound to a command buffer during a subpass **must** match. If set to `VK_TRUE`, the implementation supports variable sample locations in a subpass. If set to `VK_FALSE`, then the sample locations **must** stay constant in each subpass.

If the `VkPhysicalDeviceSampleLocationsPropertiesEXT` structure is included in the pNext chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSampleLocationsPropertiesEXT-sType-sType
sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLE_LOCATIONS_PROPERTIES_EXT

The `VkPhysicalDeviceExternalMemoryHostPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_external_memory_host
typedef struct VkPhysicalDeviceExternalMemoryHostPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize        minImportedHostPointerAlignment;
} VkPhysicalDeviceExternalMemoryHostPropertiesEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `minImportedHostPointerAlignment` is the minimum **required** alignment, in bytes, for the base address and size of host pointers that **can** be imported to a Vulkan memory object. The value **must** be a power of two.

If the `VkPhysicalDeviceExternalMemoryHostPropertiesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalMemoryHostPropertiesEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_HOST_PROPERTIES_EXT`

The `VkPhysicalDevicePointClippingProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDevicePointClippingProperties {
    VkStructureType    sType;
    void*              pNext;
    VkPointClippingBehavior    pointClippingBehavior;
} VkPhysicalDevicePointClippingProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pointClippingBehavior` is a `VkPointClippingBehavior` value specifying the point clipping behavior supported by the implementation.

If the `VkPhysicalDevicePointClippingProperties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- VUID-VkPhysicalDevicePointClippingProperties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_POINT_CLIPPING_PROPERTIES`

The `VkPhysicalDeviceSubgroupProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceSubgroupProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           subgroupSize;
    VkShaderStageFlags supportedStages;
    VkSubgroupFeatureFlags supportedOperations;
    VkBool32           quadOperationsInAllStages;
} VkPhysicalDeviceSubgroupProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `subgroupSize` is the default number of invocations in each subgroup. `subgroupSize` is at least 1 if any of the physical device's queues support `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`. `subgroupSize` is a power-of-two.
- `supportedStages` is a bitfield of `VkShaderStageFlagBits` describing the shader stages that `group operations` with `subgroup scope` are supported in. `supportedStages` will have the `VK_SHADER_STAGE_COMPUTE_BIT` bit set if any of the physical device's queues support `VK_QUEUE_COMPUTE_BIT`.
- `supportedOperations` is a bitmask of `VkSubgroupFeatureFlagBits` specifying the sets of `group operations` with `subgroup scope` supported on this device. `supportedOperations` will have the `VK_SUBGROUP_FEATURE_BASIC_BIT` bit set if any of the physical device's queues support `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`.
- `quadOperationsInAllStages` is a boolean specifying whether `quad group operations` are available in all stages, or are restricted to fragment and compute stages.

If the `VkPhysicalDeviceSubgroupProperties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

If `supportedOperations` includes `VK_SUBGROUP_FEATURE_QUAD_BIT`, `subgroupSize` **must** be greater than or equal to 4.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSubgroupProperties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_PROPERTIES`

Bits which **can** be set in `VkPhysicalDeviceSubgroupProperties::supportedOperations` and `VkPhysicalDeviceVulkan11Properties::subgroupSupportedOperations` to specify supported `group operations` with `subgroup scope` are:


```
// Provided by VK_VERSION_1_1
typedef enum VkSubgroupFeatureFlagBits {
    VK_SUBGROUP_FEATURE_BASIC_BIT = 0x00000001,
    VK_SUBGROUP_FEATURE_VOTE_BIT = 0x00000002,
    VK_SUBGROUP_FEATURE_ARITHMETIC_BIT = 0x00000004,
    VK_SUBGROUP_FEATURE_BALLOT_BIT = 0x00000008,
    VK_SUBGROUP_FEATURE_SHUFFLE_BIT = 0x00000010,
    VK_SUBGROUP_FEATURE_SHUFFLE_RELATIVE_BIT = 0x00000020,
    VK_SUBGROUP_FEATURE_CLUSTERED_BIT = 0x00000040,
    VK_SUBGROUP_FEATURE_QUAD_BIT = 0x00000080,
} VkSubgroupFeatureFlagBits;
```

- `VK_SUBGROUP_FEATURE_BASIC_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniform` capability.
- `VK_SUBGROUP_FEATURE_VOTE_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformVote` capability.
- `VK_SUBGROUP_FEATURE_ARITHMETIC_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformArithmetic` capability.
- `VK_SUBGROUP_FEATURE_BALLOT_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformBallot` capability.
- `VK_SUBGROUP_FEATURE_SHUFFLE_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformShuffle` capability.
- `VK_SUBGROUP_FEATURE_SHUFFLE_RELATIVE_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformShuffleRelative` capability.
- `VK_SUBGROUP_FEATURE_CLUSTERED_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformClustered` capability.
- `VK_SUBGROUP_FEATURE_QUAD_BIT` specifies the device will accept SPIR-V shader modules containing the `GroupNonUniformQuad` capability.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkSubgroupFeatureFlags;
```

`VkSubgroupFeatureFlags` is a bitmask type for setting a mask of zero or more `VkSubgroupFeatureFlagBits`.

The `VkPhysicalDeviceSubgroupSizeControlProperties` structure is defined as:

```
typedef struct VkPhysicalDeviceSubgroupSizeControlProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           minSubgroupSize;
    uint32_t           maxSubgroupSize;
    uint32_t           maxComputeWorkgroupSubgroups;
    VkShaderStageFlags requiredSubgroupSizeStages;
} VkPhysicalDeviceSubgroupSizeControlProperties;
```

```
// Provided by VK_EXT_subgroup_size_control
typedef VkPhysicalDeviceSubgroupSizeControlProperties
VkPhysicalDeviceSubgroupSizeControlPropertiesEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `minSubgroupSize` is the minimum subgroup size supported by this device. `minSubgroupSize` is at least one if any of the physical device's queues support `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`. `minSubgroupSize` is a power-of-two. `minSubgroupSize` is less than or equal to `maxSubgroupSize`. `minSubgroupSize` is less than or equal to `subgroupSize`.
- `maxSubgroupSize` is the maximum subgroup size supported by this device. `maxSubgroupSize` is at least one if any of the physical device's queues support `VK_QUEUE_GRAPHICS_BIT` or `VK_QUEUE_COMPUTE_BIT`. `maxSubgroupSize` is a power-of-two. `maxSubgroupSize` is greater than or equal to `minSubgroupSize`. `maxSubgroupSize` is greater than or equal to `subgroupSize`.
- `maxComputeWorkgroupSubgroups` is the maximum number of subgroups supported by the implementation within a workgroup.
- `requiredSubgroupSizeStages` is a bitfield of what shader stages support having a required subgroup size specified.

If the `VkPhysicalDeviceSubgroupSizeControlProperties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

If `VkPhysicalDeviceSubgroupProperties::supportedOperations` includes `VK_SUBGROUP_FEATURE_QUAD_BIT`, `minSubgroupSize` **must** be greater than or equal to 4.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSubgroupSizeControlProperties-sType-sType `sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_PROPERTIES`

The `VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_blend_operation_advanced
typedef struct VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           advancedBlendMaxColorAttachments;
    VkBool32           advancedBlendIndependentBlend;
    VkBool32           advancedBlendNonPremultipliedSrcColor;
    VkBool32           advancedBlendNonPremultipliedDstColor;
    VkBool32           advancedBlendCorrelatedOverlap;
    VkBool32           advancedBlendAllOperations;
} VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `advancedBlendMaxColorAttachments` is one greater than the highest color attachment index that **can** be used in a render pass instance, for a pipeline that uses an [advanced blend operation](#).
- `advancedBlendIndependentBlend` specifies whether advanced blend operations **can** vary per-attachment.
- `advancedBlendNonPremultipliedSrcColor` specifies whether the source color **can** be treated as non-premultiplied. If this is `VK_FALSE`, then `VkPipelineColorBlendAdvancedStateCreateInfoEXT::srcPremultiplied` **must** be `VK_TRUE`.
- `advancedBlendNonPremultipliedDstColor` specifies whether the destination color **can** be treated as non-premultiplied. If this is `VK_FALSE`, then `VkPipelineColorBlendAdvancedStateCreateInfoEXT::dstPremultiplied` **must** be `VK_TRUE`.
- `advancedBlendCorrelatedOverlap` specifies whether the overlap mode **can** be treated as correlated. If this is `VK_FALSE`, then `VkPipelineColorBlendAdvancedStateCreateInfoEXT::blendOverlap` **must** be `VK_BLEND_OVERLAP_UNCORRELATED_EXT`.
- `advancedBlendAllOperations` specifies whether all advanced blend operation enums are supported. See the valid usage of `VkPipelineColorBlendAttachmentState`.

If the `VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT-sType-sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BLEND_OPERATION_ADVANCED_PROPERTIES_EXT`

The `VkPhysicalDeviceVertexAttributeDivisorProperties` structure is defined as:

```
typedef struct VkPhysicalDeviceVertexAttributeDivisorProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxVertexAttribDivisor;
    VkBool32           supportsNonZeroFirstInstance;
} VkPhysicalDeviceVertexAttributeDivisorProperties;
```

```
// Provided by VK_KHR_vertex_attribute_divisor
typedef VkPhysicalDeviceVertexAttributeDivisorProperties
VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxVertexAttribDivisor` is the maximum value of the number of instances that will repeat the value of vertex attribute data when instanced rendering is enabled.
- `supportsNonZeroFirstInstance` specifies whether a non-zero value for the `firstInstance` parameter of `drawing commands` is supported when `VkVertexInputBindingDivisorDescription::divisor` is not 1.

If the `VkPhysicalDeviceVertexAttributeDivisorProperties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceVertexAttributeDivisorProperties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_PROPERTIES`

The `VkPhysicalDeviceSamplerFilterMinmaxProperties` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceSamplerFilterMinmaxProperties {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           filterMinmaxSingleComponentFormats;
    VkBool32           filterMinmaxImageComponentMapping;
} VkPhysicalDeviceSamplerFilterMinmaxProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `filterMinmaxSingleComponentFormats` is a boolean value indicating whether a minimum set of required formats support min/max filtering.

- `filterMinmaxImageComponentMapping` is a boolean value indicating whether the implementation supports non-identity component mapping of the image when doing min/max filtering.

If the `VkPhysicalDeviceSamplerFilterMinmaxProperties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

If `filterMinmaxSingleComponentFormats` is `VK_TRUE`, the following formats **must** support the `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT` feature with `VK_IMAGE_TILING_OPTIMAL`, if they support `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`:

- `VK_FORMAT_R8_UNORM`
- `VK_FORMAT_R8_SNORM`
- `VK_FORMAT_R16_UNORM`
- `VK_FORMAT_R16_SNORM`
- `VK_FORMAT_R16_SFLOAT`
- `VK_FORMAT_R32_SFLOAT`
- `VK_FORMAT_D16_UNORM`
- `VK_FORMAT_X8_D24_UNORM_PACK32`
- `VK_FORMAT_D32_SFLOAT`
- `VK_FORMAT_D16_UNORM_S8_UINT`
- `VK_FORMAT_D24_UNORM_S8_UINT`
- `VK_FORMAT_D32_SFLOAT_S8_UINT`

If the format is a depth/stencil format, this bit only specifies that the depth aspect (not the stencil aspect) of an image of this format supports min/max filtering, and that min/max filtering of the depth aspect is supported when depth compare is disabled in the sampler.

If `filterMinmaxImageComponentMapping` is `VK_FALSE` the component mapping of the image view used with min/max filtering **must** have been created with the `r` component set to the `identity swizzle`. Only the `r` component of the sampled image value is defined and the other component values are undefined. If `filterMinmaxImageComponentMapping` is `VK_TRUE` this restriction does not apply and image component mapping works as normal.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceSamplerFilterMinmaxProperties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_FILTER_MINMAX_PROPERTIES`

The `VkPhysicalDeviceProtectedMemoryProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceProtectedMemoryProperties {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           protectedNoFault;
} VkPhysicalDeviceProtectedMemoryProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `protectedNoFault` specifies how an implementation behaves when an application attempts to write to unprotected memory in a protected queue operation, read from protected memory in an unprotected queue operation, or perform a query in a protected queue operation. If this limit is `VK_TRUE`, such writes will be discarded or have undefined values written, reads and queries will return undefined values. If this limit is `VK_FALSE`, applications **must** not perform these operations. See [Protected Memory Access Rules](#) for more information.

If the `VkPhysicalDeviceProtectedMemoryProperties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceProtectedMemoryProperties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROTECTED_MEMORY_PROPERTIES`

The `VkPhysicalDeviceMaintenance3Properties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceMaintenance3Properties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxPerSetDescriptors;
    VkDeviceSize        maxMemoryAllocationSize;
} VkPhysicalDeviceMaintenance3Properties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxPerSetDescriptors` is a maximum number of descriptors (summed over all descriptor types) in a single descriptor set that is guaranteed to satisfy any implementation-dependent constraints on the size of a descriptor set itself. Applications **can** query whether a descriptor set that goes beyond this limit is supported using `vkGetDescriptorSetLayoutSupport`.
- `maxMemoryAllocationSize` is the maximum size of a memory allocation that **can** be created, even if there is more space available in the heap. If `VkMemoryAllocateInfo::allocationSize` is larger

the error `VK_ERROR_OUT_OF_DEVICE_MEMORY` **may** be returned.

If the `VkPhysicalDeviceMaintenance3Properties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceMaintenance3Properties-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_3_PROPERTIES`

The `VkPhysicalDeviceDescriptorIndexingProperties` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceDescriptorIndexingProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxUpdateAfterBindDescriptorsInAllPools;
    VkBool32           shaderUniformBufferArrayNonUniformIndexingNative;
    VkBool32           shaderSampledImageArrayNonUniformIndexingNative;
    VkBool32           shaderStorageBufferArrayNonUniformIndexingNative;
    VkBool32           shaderStorageImageArrayNonUniformIndexingNative;
    VkBool32           shaderInputAttachmentArrayNonUniformIndexingNative;
    VkBool32           robustBufferAccessUpdateAfterBind;
    VkBool32           quadDivergentImplicitLod;
    uint32_t           maxPerStageDescriptorUpdateAfterBindSamplers;
    uint32_t           maxPerStageDescriptorUpdateAfterBindUniformBuffers;
    uint32_t           maxPerStageDescriptorUpdateAfterBindStorageBuffers;
    uint32_t           maxPerStageDescriptorUpdateAfterBindSampledImages;
    uint32_t           maxPerStageDescriptorUpdateAfterBindStorageImages;
    uint32_t           maxPerStageDescriptorUpdateAfterBindInputAttachments;
    uint32_t           maxPerStageUpdateAfterBindResources;
    uint32_t           maxDescriptorSetUpdateAfterBindSamplers;
    uint32_t           maxDescriptorSetUpdateAfterBindUniformBuffers;
    uint32_t           maxDescriptorSetUpdateAfterBindUniformBuffersDynamic;
    uint32_t           maxDescriptorSetUpdateAfterBindStorageBuffers;
    uint32_t           maxDescriptorSetUpdateAfterBindStorageBuffersDynamic;
    uint32_t           maxDescriptorSetUpdateAfterBindSampledImages;
    uint32_t           maxDescriptorSetUpdateAfterBindStorageImages;
    uint32_t           maxDescriptorSetUpdateAfterBindInputAttachments;
} VkPhysicalDeviceDescriptorIndexingProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxUpdateAfterBindDescriptorsInAllPools` is the maximum number of descriptors (summed over all descriptor types) that **can** be created across all pools that are created with the `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT` bit set. Pool creation **may** fail when this limit

is exceeded, or when the space this limit represents is unable to satisfy a pool creation due to fragmentation.

- `shaderUniformBufferArrayNonUniformIndexingNative` is a boolean value indicating whether uniform buffer descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of uniform buffers **may** execute multiple times in order to access all the descriptors.
- `shaderSampledImageArrayNonUniformIndexingNative` is a boolean value indicating whether sampler and image descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of samplers or images **may** execute multiple times in order to access all the descriptors.
- `shaderStorageBufferArrayNonUniformIndexingNative` is a boolean value indicating whether storage buffer descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of storage buffers **may** execute multiple times in order to access all the descriptors.
- `shaderStorageImageArrayNonUniformIndexingNative` is a boolean value indicating whether storage image descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of storage images **may** execute multiple times in order to access all the descriptors.
- `shaderInputAttachmentArrayNonUniformIndexingNative` is a boolean value indicating whether input attachment descriptors natively support non-uniform indexing. If this is `VK_FALSE`, then a single dynamic instance of an instruction that non-uniformly indexes an array of input attachments **may** execute multiple times in order to access all the descriptors.
- `robustBufferAccessUpdateAfterBind` is a boolean value indicating whether `robustBufferAccess` **can** be enabled on a device simultaneously with `descriptorBindingUniformBufferUpdateAfterBind`, `descriptorBindingStorageBufferUpdateAfterBind`, `descriptorBindingUniformTexelBufferUpdateAfterBind`, and/or `descriptorBindingStorageTexelBufferUpdateAfterBind`. If this is `VK_FALSE`, then either `robustBufferAccess` **must** be disabled or all of these update-after-bind features **must** be disabled.
- `quadDivergentImplicitLod` is a boolean value indicating whether implicit LOD calculations for image operations have well-defined results when the image and/or sampler objects used for the instruction are not uniform within a quad. See [Derivative Image Operations](#).
- `maxPerStageDescriptorUpdateAfterBindSamplers` is similar to `maxPerStageDescriptorSamplers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindUniformBuffers` is similar to `maxPerStageDescriptorUniformBuffers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindStorageBuffers` is similar to `maxPerStageDescriptorStorageBuffers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindSampledImages` is similar to `maxPerStageDescriptorSampledImages` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.

- `maxPerStageDescriptorUpdateAfterBindStorageImages` is similar to `maxPerStageDescriptorStorageImages` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageDescriptorUpdateAfterBindInputAttachments` is similar to `maxPerStageDescriptorInputAttachments` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxPerStageUpdateAfterBindResources` is similar to `maxPerStageResources` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxDescriptorSetUpdateAfterBindSamplers` is similar to `maxDescriptorSetSamplers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxDescriptorSetUpdateAfterBindUniformBuffers` is similar to `maxDescriptorSetUniformBuffers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxDescriptorSetUpdateAfterBindUniformBuffersDynamic` is similar to `maxDescriptorSetUniformBuffersDynamic` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set. While an application **can** allocate dynamic uniform buffer descriptors from a pool created with the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT`, bindings for these descriptors **must** not be present in any descriptor set layout that includes bindings created with `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`.
- `maxDescriptorSetUpdateAfterBindStorageBuffers` is similar to `maxDescriptorSetStorageBuffers` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxDescriptorSetUpdateAfterBindStorageBuffersDynamic` is similar to `maxDescriptorSetStorageBuffersDynamic` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set. While an application **can** allocate dynamic storage buffer descriptors from a pool created with the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT`, bindings for these descriptors **must** not be present in any descriptor set layout that includes bindings created with `VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT`.
- `maxDescriptorSetUpdateAfterBindSampledImages` is similar to `maxDescriptorSetSampledImages` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxDescriptorSetUpdateAfterBindStorageImages` is similar to `maxDescriptorSetStorageImages` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.
- `maxDescriptorSetUpdateAfterBindInputAttachments` is similar to `maxDescriptorSetInputAttachments` but counts descriptors from descriptor sets created with or without the `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT` bit set.

If the `VkPhysicalDeviceDescriptorIndexingProperties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with

each corresponding implementation-dependent property.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceDescriptorIndexingProperties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_PROPERTIES`

The `VkPhysicalDeviceConservativeRasterizationPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_conservative_rasterization
typedef struct VkPhysicalDeviceConservativeRasterizationPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    float               primitiveOverestimationSize;
    float               maxExtraPrimitiveOverestimationSize;
    float               extraPrimitiveOverestimationSizeGranularity;
    VkBool32            primitiveUnderestimation;
    VkBool32            conservativePointAndLineRasterization;
    VkBool32            degenerateTrianglesRasterized;
    VkBool32            degenerateLinesRasterized;
    VkBool32            fullyCoveredFragmentShaderInputVariable;
    VkBool32            conservativeRasterizationPostDepthCoverage;
} VkPhysicalDeviceConservativeRasterizationPropertiesEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `primitiveOverestimationSize` is the size in pixels the generating primitive is increased at each of its edges during conservative rasterization overestimation mode. Even with a size of 0.0, conservative rasterization overestimation rules still apply and if any part of the pixel rectangle is covered by the generating primitive, fragments are generated for the entire pixel. However implementations **may** make the pixel coverage area even more conservative by increasing the size of the generating primitive.
- `maxExtraPrimitiveOverestimationSize` is the maximum size in pixels of extra overestimation the implementation supports in the pipeline state. A value of 0.0 means the implementation does not support any additional overestimation of the generating primitive during conservative rasterization. A value above 0.0 allows the application to further increase the size of the generating primitive during conservative rasterization overestimation.
- `extraPrimitiveOverestimationSizeGranularity` is the granularity of extra overestimation that can be specified in the pipeline state between 0.0 and `maxExtraPrimitiveOverestimationSize` inclusive. A value of 0.0 means the implementation can use the smallest representable non-zero value in the screen space pixel fixed-point grid.
- `primitiveUnderestimation` is `VK_TRUE` if the implementation supports the `VK_CONSERVATIVE_RASTERIZATION_MODE_UNDERESTIMATE_EXT` conservative rasterization mode in addition to `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT`. Otherwise the implementation only supports `VK_CONSERVATIVE_RASTERIZATION_MODE_OVERESTIMATE_EXT`.

- `conservativePointAndLineRasterization` is `VK_TRUE` if the implementation supports conservative rasterization of point and line primitives as well as triangle primitives. Otherwise the implementation only supports triangle primitives.
- `degenerateTrianglesRasterized` is `VK_FALSE` if the implementation culls primitives generated from triangles that become zero area after they are quantized to the fixed-point rasterization pixel grid. `degenerateTrianglesRasterized` is `VK_TRUE` if these primitives are not culled and the provoking vertex attributes and depth value are used for the fragments. The primitive area calculation is done on the primitive generated from the clipped triangle if applicable. Zero area primitives are backfacing and the application **can** enable backface culling if desired.
- `degenerateLinesRasterized` is `VK_FALSE` if the implementation culls lines that become zero length after they are quantized to the fixed-point rasterization pixel grid. `degenerateLinesRasterized` is `VK_TRUE` if zero length lines are not culled and the provoking vertex attributes and depth value are used for the fragments.
- `fullyCoveredFragmentShaderInputVariable` is `VK_TRUE` if the implementation supports the SPIR-V builtin fragment shader input variable `FullyCoveredEXT` specifying that conservative rasterization is enabled and the fragment area is fully covered by the generating primitive.
- `conservativeRasterizationPostDepthCoverage` is `VK_TRUE` if the implementation supports conservative rasterization with the `PostDepthCoverage` execution mode enabled. Otherwise the `PostDepthCoverage` execution mode **must** not be used when conservative rasterization is enabled.

If the `VkPhysicalDeviceConservativeRasterizationPropertiesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceConservativeRasterizationPropertiesEXT-sType-sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CONSERVATIVE_RASTERIZATION_PROPERTIES_EXT`

The `VkPhysicalDeviceDepthStencilResolveProperties` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceDepthStencilResolveProperties {
    VkStructureType    sType;
    void*              pNext;
    VkResolveModeFlags supportedDepthResolveModes;
    VkResolveModeFlags supportedStencilResolveModes;
    VkBool32           independentResolveNone;
    VkBool32           independentResolve;
} VkPhysicalDeviceDepthStencilResolveProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.

- `supportedDepthResolveModes` is a bitmask of `VkResolveModeFlagBits` indicating the set of supported depth resolve modes. A value of `VK_RESOLVE_MODE_NONE` indicates that depth resolve operations are disallowed [SCID-8]. If any bits are set then `VK_RESOLVE_MODE_SAMPLE_ZERO_BIT` **must** be included in the set but implementations **may** support additional modes.
- `supportedStencilResolveModes` is a bitmask of `VkResolveModeFlagBits` indicating the set of supported stencil resolve modes. A value of `VK_RESOLVE_MODE_NONE` indicates that stencil resolve operations are disallowed [SCID-8]. If any bits are set then `VK_RESOLVE_MODE_SAMPLE_ZERO_BIT` **must** be included in the set but implementations **may** support additional modes. `VK_RESOLVE_MODE_AVERAGE_BIT` **must** not be included in the set.
- `independentResolveNone` is `VK_TRUE` if the implementation supports setting the depth and stencil resolve modes to different values when one of those modes is `VK_RESOLVE_MODE_NONE`. Otherwise the implementation only supports setting both modes to the same value.
- `independentResolve` is `VK_TRUE` if the implementation supports all combinations of the supported depth and stencil resolve modes, including setting either depth or stencil resolve mode to `VK_RESOLVE_MODE_NONE`. An implementation that supports `independentResolve` **must** also support `independentResolveNone`.

If the `VkPhysicalDeviceDepthStencilResolveProperties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- `VUID-VkPhysicalDeviceDepthStencilResolveProperties-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_STENCIL_RESOLVE_PROPERTIES`

The `VkPhysicalDevicePerformanceQueryPropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_performance_query
typedef struct VkPhysicalDevicePerformanceQueryPropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           allowCommandBufferQueryCopies;
} VkPhysicalDevicePerformanceQueryPropertiesKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `allowCommandBufferQueryCopies` is `VK_TRUE` if the performance query pools are allowed to be used with `vkCmdCopyQueryPoolResults`.

If the `VkPhysicalDevicePerformanceQueryPropertiesKHR` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- VUID-VkPhysicalDevicePerformanceQueryPropertiesKHR-sType-sType
sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PERFORMANCE_QUERY_PROPERTIES_KHR

The `VkPhysicalDeviceTexelBufferAlignmentProperties` structure is defined as:

```
typedef struct VkPhysicalDeviceTexelBufferAlignmentProperties {  
    VkStructureType    sType;  
    void*              pNext;  
    VkDeviceSize       storageTexelBufferOffsetAlignmentBytes;  
    VkBool32           storageTexelBufferOffsetSingleTexelAlignment;  
    VkDeviceSize       uniformTexelBufferOffsetAlignmentBytes;  
    VkBool32           uniformTexelBufferOffsetSingleTexelAlignment;  
} VkPhysicalDeviceTexelBufferAlignmentProperties;
```

```
// Provided by VK_EXT_texel_buffer_alignment  
typedef VkPhysicalDeviceTexelBufferAlignmentProperties  
VkPhysicalDeviceTexelBufferAlignmentPropertiesEXT;
```

- sType is a `VkStructureType` value identifying this structure.
- pNext is `NULL` or a pointer to a structure extending this structure.
- storageTexelBufferOffsetAlignmentBytes is a byte alignment that is sufficient for a storage texel buffer of any format. The value **must** be a power of two.
- storageTexelBufferOffsetSingleTexelAlignment indicates whether single texel alignment is sufficient for a storage texel buffer of any format.
- uniformTexelBufferOffsetAlignmentBytes is a byte alignment that is sufficient for a uniform texel buffer of any format. The value **must** be a power of two.
- uniformTexelBufferOffsetSingleTexelAlignment indicates whether single texel alignment is sufficient for a uniform texel buffer of any format.

If the `VkPhysicalDeviceTexelBufferAlignmentProperties` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

If the single texel alignment property is `VK_FALSE`, then the buffer view's offset **must** be aligned to the corresponding byte alignment value. If the single texel alignment property is `VK_TRUE`, then the buffer view's offset **must** be aligned to the lesser of the corresponding byte alignment value or the size of a single texel, based on `VkBufferViewCreateInfo::format`. If the size of a single texel is a multiple of three bytes, then the size of a single component of the format is used instead.

These limits **must** not advertise a larger alignment than the `required` maximum minimum value of `VkPhysicalDeviceLimits::minTexelBufferOffsetAlignment`, for any format that supports use as a texel buffer.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceTexelBufferAlignmentProperties-sType-sType
sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_PROPERTIES

The `VkPhysicalDeviceTimelineSemaphoreProperties` structure is defined as:

```
// Provided by VK_VERSION_1_2
typedef struct VkPhysicalDeviceTimelineSemaphoreProperties {
    VkStructureType    sType;
    void*              pNext;
    uint64_t           maxTimelineSemaphoreValueDifference;
} VkPhysicalDeviceTimelineSemaphoreProperties;
```

- sType is a `VkStructureType` value identifying this structure.
- pNext is `NULL` or a pointer to a structure extending this structure.
- maxTimelineSemaphoreValueDifference indicates the maximum difference allowed by the implementation between the current value of a timeline semaphore and any pending signal or wait operations.

If the `VkPhysicalDeviceTimelineSemaphoreProperties` structure is included in the pNext chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceTimelineSemaphoreProperties-sType-sType
sType **must** be VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_PROPERTIES

The `VkPhysicalDeviceLineRasterizationProperties` structure is defined as:

```
typedef struct VkPhysicalDeviceLineRasterizationProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           lineSubPixelPrecisionBits;
} VkPhysicalDeviceLineRasterizationProperties;
```

```
// Provided by VK_KHR_line_rasterization
typedef VkPhysicalDeviceLineRasterizationProperties
VkPhysicalDeviceLineRasterizationPropertiesKHR;
```



```
// Provided by VK_EXT_line_rasterization
typedef VkPhysicalDeviceLineRasterizationProperties
VkPhysicalDeviceLineRasterizationPropertiesEXT;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **lineSubPixelPrecisionBits** is the number of bits of subpixel precision in framebuffer coordinates x_f and y_f when rasterizing [line segments](#).

If the [VkPhysicalDeviceLineRasterizationProperties](#) structure is included in the **pNext** chain of the [VkPhysicalDeviceProperties2](#) structure passed to [vkGetPhysicalDeviceProperties2](#), it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceLineRasterizationProperties-sType-sType **sType must** be [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES](#)

The [VkPhysicalDeviceRobustness2PropertiesKHR](#) structure is defined as:

```
typedef struct VkPhysicalDeviceRobustness2PropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkDeviceSize        robustStorageBufferAccessSizeAlignment;
    VkDeviceSize        robustUniformBufferAccessSizeAlignment;
} VkPhysicalDeviceRobustness2PropertiesKHR;
```

```
// Provided by VK_EXT_robustness2
typedef VkPhysicalDeviceRobustness2PropertiesKHR
VkPhysicalDeviceRobustness2PropertiesEXT;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **robustStorageBufferAccessSizeAlignment** is the number of bytes that the range of a storage buffer descriptor is rounded up to when used for bounds-checking when the [robustBufferAccess2](#) feature is enabled. This value **must** be either 1 or 4.
- **robustUniformBufferAccessSizeAlignment** is the number of bytes that the range of a uniform buffer descriptor is rounded up to when used for bounds-checking when the [robustBufferAccess2](#) feature is enabled. This value **must** be a power of two in the range [1, 256].

If the [VkPhysicalDeviceRobustness2PropertiesKHR](#) structure is included in the **pNext** chain of the [VkPhysicalDeviceProperties2](#) structure passed to [vkGetPhysicalDeviceProperties2](#), it is filled in with

each corresponding implementation-dependent property.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceRobustness2PropertiesKHR-sType-sType **sType must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_PROPERTIES_KHR`

The `VkPhysicalDeviceFragmentShadingRatePropertiesKHR` structure is defined as:

```
// Provided by VK_KHR_fragment_shading_rate
typedef struct VkPhysicalDeviceFragmentShadingRatePropertiesKHR {
    VkStructureType    sType;
    void*              pNext;
    VkExtent2D         minFragmentShadingRateAttachmentTexelSize;
    VkExtent2D         maxFragmentShadingRateAttachmentTexelSize;
    uint32_t           maxFragmentShadingRateAttachmentTexelSizeAspectRatio;
    VkBool32           primitiveFragmentShadingRateWithMultipleViewports;
    VkBool32           layeredShadingRateAttachments;
    VkBool32           fragmentShadingRateNonTrivialCombinerOps;
    VkExtent2D         maxFragmentSize;
    uint32_t           maxFragmentSizeAspectRatio;
    uint32_t           maxFragmentShadingRateCoverageSamples;
    VkSampleCountFlagBits maxFragmentShadingRateRasterizationSamples;
    VkBool32           fragmentShadingRateWithShaderDepthStencilWrites;
    VkBool32           fragmentShadingRateWithSampleMask;
    VkBool32           fragmentShadingRateWithShaderSampleMask;
    VkBool32           fragmentShadingRateWithConservativeRasterization;
    VkBool32           fragmentShadingRateWithFragmentShaderInterlock;
    VkBool32           fragmentShadingRateWithCustomSampleLocations;
    VkBool32           fragmentShadingRateStrictMultiplyCombiner;
} VkPhysicalDeviceFragmentShadingRatePropertiesKHR;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `minFragmentShadingRateAttachmentTexelSize` indicates minimum supported width and height of the portion of the framebuffer corresponding to each texel in a fragment shading rate attachment. Each value **must** be less than or equal to the values in `maxFragmentShadingRateAttachmentTexelSize`. Each value **must** be a power-of-two. It **must** be (0,0) if the `attachmentFragmentShadingRate` feature is not supported.
- `maxFragmentShadingRateAttachmentTexelSize` indicates maximum supported width and height of the portion of the framebuffer corresponding to each texel in a fragment shading rate attachment. Each value **must** be greater than or equal to the values in `minFragmentShadingRateAttachmentTexelSize`. Each value **must** be a power-of-two. It **must** be (0,0) if the `attachmentFragmentShadingRate` feature is not supported.
- `maxFragmentShadingRateAttachmentTexelSizeAspectRatio` indicates the maximum ratio between the width and height of the portion of the framebuffer corresponding to each texel in a

fragment shading rate attachment. `maxFragmentShadingRateAttachmentTexelSizeAspectRatio` **must** be a power-of-two value, and **must** be less than or equal to $\max(\text{maxFragmentShadingRateAttachmentTexelSize.width} / \text{minFragmentShadingRateAttachmentTexelSize.height}, \text{maxFragmentShadingRateAttachmentTexelSize.height} / \text{minFragmentShadingRateAttachmentTexelSize.width})$. It **must** be 0 if the `attachmentFragmentShadingRate` feature is not supported.

- `primitiveFragmentShadingRateWithMultipleViewports` specifies whether the `primitive fragment shading rate` **can** be used when multiple viewports are used. If this value is `VK_FALSE`, only a single viewport **must** be used, and applications **must** not write to the `ViewportIndex` built-in when setting `PrimitiveShadingRateKHR`. It **must** be `VK_FALSE` if the `shaderOutputViewportIndex` feature, or the `geometryShader` feature is not supported, or if the `primitiveFragmentShadingRate` feature is not supported.
- `layeredShadingRateAttachments` specifies whether a shading rate attachment image view **can** be created with multiple layers. If this value is `VK_FALSE`, when creating an image view with a `usage` that includes `VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`, `layerCount` **must** be 1. It **must** be `VK_FALSE` if the `multiview` feature, the `shaderOutputViewportIndex` feature, or the `geometryShader` feature is not supported, or if the `attachmentFragmentShadingRate` feature is not supported.
- `fragmentShadingRateNonTrivialCombinerOps` specifies whether `VkFragmentShadingRateCombinerOpKHR` `enums` other than `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_KEEP_KHR` or `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_REPLACE_KHR` **can** be used. It **must** be `VK_FALSE` unless either the `primitiveFragmentShadingRate` or `attachmentFragmentShadingRate` feature is supported.
- `maxFragmentSize` indicates the maximum supported width and height of a fragment. Its `width` and `height` members **must** both be power-of-two values. This limit is purely informational, and is not validated.
- `maxFragmentSizeAspectRatio` indicates the maximum ratio between the width and height of a fragment. `maxFragmentSizeAspectRatio` **must** be a power-of-two value, and **must** be less than or equal to the maximum of the `width` and `height` members of `maxFragmentSize`. This limit is purely informational, and is not validated.
- `maxFragmentShadingRateCoverageSamples` specifies the maximum number of coverage samples supported in a single fragment. `maxFragmentShadingRateCoverageSamples` **must** be less than or equal to the product of the `width` and `height` members of `maxFragmentSize`, and the sample count reported by `maxFragmentShadingRateRasterizationSamples`. `maxFragmentShadingRateCoverageSamples` **must** be less than or equal to `maxSampleMaskWords` × 32 if `fragmentShadingRateWithShaderSampleMask` is supported. This limit is purely informational, and is not validated.
- `maxFragmentShadingRateRasterizationSamples` is a `VkSampleCountFlagBits` value specifying the maximum sample rate supported when a fragment covers multiple pixels. This limit is purely informational, and is not validated.
- `fragmentShadingRateWithShaderDepthStencilWrites` specifies whether the implementation supports writing `FragDepth` or `FragStencilRefEXT` from a fragment shader for multi-pixel fragments. If this value is `VK_FALSE`, writing to those built-ins will clamp the fragment shading rate to (1,1).

- `fragmentShadingRateWithSampleMask` specifies whether the implementation supports setting valid bits of `VkPipelineMultisampleStateCreateInfo::pSampleMask` to 0 for multi-pixel fragments. If this value is `VK_FALSE`, zeroing valid bits in the sample mask will clamp the fragment shading rate to (1,1).
- `fragmentShadingRateWithShaderSampleMask` specifies whether the implementation supports reading or writing `SampleMask` for multi-pixel fragments. If this value is `VK_FALSE`, using that built-in will clamp the fragment shading rate to (1,1).
- `fragmentShadingRateWithConservativeRasterization` specifies whether `conservative rasterization` is supported for multi-pixel fragments. It **must** be `VK_FALSE` if `VK_EXT_conservative_rasterization` is not supported. If this value is `VK_FALSE`, using `conservative rasterization` will clamp the fragment shading rate to (1,1).
- `fragmentShadingRateWithFragmentShaderInterlock` specifies whether `fragment shader interlock` is supported for multi-pixel fragments. It **must** be `VK_FALSE` if `VK_EXT_fragment_shader_interlock` is not supported. If this value is `VK_FALSE`, using `fragment shader interlock` will clamp the fragment shading rate to (1,1).
- `fragmentShadingRateWithCustomSampleLocations` specifies whether `custom sample locations` are supported for multi-pixel fragments. It **must** be `VK_FALSE` if `VK_EXT_sample_locations` is not supported. If this value is `VK_FALSE`, using `custom sample locations` will clamp the fragment shading rate to (1,1).
- `fragmentShadingRateStrictMultiplyCombiner` specifies whether `VK_FRAGMENT_SHADING_RATE_COMBINER_OP_MUL_KHR` accurately performs a multiplication or not. Implementations where this value is `VK_FALSE` will instead combine rates with an addition. If `fragmentShadingRateNonTrivialCombinerOps` is `VK_FALSE`, implementations **must** report this as `VK_FALSE`. If `fragmentShadingRateNonTrivialCombinerOps` is `VK_TRUE`, implementations **should** report this as `VK_TRUE`.

NOTE

Multiplication of the combiner rates using the fragment width/height in linear space is equivalent to an addition of those values in log2 space. Some implementations inadvertently implemented an addition in linear space due to unclear requirements originating outside of this specification. This resulted in `fragmentShadingRateStrictMultiplyCombiner` being added. Fortunately, this only affects situations where a rate of 1 in either dimension is combined with another rate of 1. All other combinations result in the exact same result as if multiplication was performed in linear space due to the clamping logic, and the fact that both the sum and product of 2 and 2 are equal. In many cases, this limit will not affect the correct operation of applications.

If the `VkPhysicalDeviceFragmentShadingRatePropertiesKHR` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

These properties are related to `fragment shading rates`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceFragmentShadingRatePropertiesKHR-sType-sType

sType must be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_PROPERTIES_KHR`

The `VkPhysicalDeviceCustomBorderColorPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_custom_border_color
typedef struct VkPhysicalDeviceCustomBorderColorPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           maxCustomBorderColorSamplers;
} VkPhysicalDeviceCustomBorderColorPropertiesEXT;
```

- `maxCustomBorderColorSamplers` indicates the maximum number of samplers with custom border colors which **can** simultaneously exist on a device.

If the `VkPhysicalDeviceCustomBorderColorPropertiesEXT` structure is included in the `pNext` chain of the `VkPhysicalDeviceProperties2` structure passed to `vkGetPhysicalDeviceProperties2`, it is filled in with each corresponding implementation-dependent property.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceCustomBorderColorPropertiesEXT-sType-sType
sType must be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUSTOM_BORDER_COLOR_PROPERTIES_EXT`

34.1. Limit Requirements

The following table specifies the **required** minimum/maximum for all Vulkan graphics implementations. Where a limit corresponds to a fine-grained device feature which is **optional**, the feature name is listed with two **required** limits, one when the feature is supported and one when it is not supported. If an implementation supports a feature, the limits reported are the same whether or not the feature is enabled.

Table 48. Required Limit Types

Type	Limit	Feature
<code>uint32_t</code>	<code>maxImageDimension1D</code>	-
<code>uint32_t</code>	<code>maxImageDimension2D</code>	-
<code>uint32_t</code>	<code>maxImageDimension3D</code>	-
<code>uint32_t</code>	<code>maxImageDimensionCube</code>	-
<code>uint32_t</code>	<code>maxImageArrayLayers</code>	-
<code>uint32_t</code>	<code>maxTexelBufferElements</code>	-
<code>uint32_t</code>	<code>maxUniformBufferRange</code>	-
<code>uint32_t</code>	<code>maxStorageBufferRange</code>	-
<code>uint32_t</code>	<code>maxPushConstantsSize</code>	-

Type	Limit	Feature
uint32_t	maxMemoryAllocationCount	-
uint32_t	maxSamplerAllocationCount	-
VkDeviceSize	bufferImageGranularity	-
VkDeviceSize	sparseAddressSpaceSize	sparseBinding
uint32_t	maxBoundDescriptorSets	-
uint32_t	maxPerStageDescriptorSamplers	-
uint32_t	maxPerStageDescriptorUniformBuffers	-
uint32_t	maxPerStageDescriptorStorageBuffers	-
uint32_t	maxPerStageDescriptorSampledImages	-
uint32_t	maxPerStageDescriptorStorageImages	-
uint32_t	maxPerStageDescriptorInputAttachments	-
uint32_t	maxPerStageResources	-
uint32_t	maxDescriptorSetSamplers	-
uint32_t	maxDescriptorSetUniformBuffers	-
uint32_t	maxDescriptorSetUniformBuffersDynamic	-
uint32_t	maxDescriptorSetStorageBuffers	-
uint32_t	maxDescriptorSetStorageBuffersDynamic	-
uint32_t	maxDescriptorSetSampledImages	-
uint32_t	maxDescriptorSetStorageImages	-
uint32_t	maxDescriptorSetInputAttachments	-
uint32_t	maxVertexInputAttributes	-
uint32_t	maxVertexInputBindings	-
uint32_t	maxVertexInputAttributeOffset	-
uint32_t	maxVertexInputBindingStride	-
uint32_t	maxVertexOutputComponents	-
uint32_t	maxTessellationGenerationLevel	tessellationShader
uint32_t	maxTessellationPatchSize	tessellationShader
uint32_t	maxTessellationControlPerVertexInputComponents	tessellationShader
uint32_t	maxTessellationControlPerVertexOutputComponents	tessellationShader
uint32_t	maxTessellationControlPerPatchOutputComponents	tessellationShader
uint32_t	maxTessellationControlTotalOutputComponents	tessellationShader
uint32_t	maxTessellationEvaluationInputComponents	tessellationShader
uint32_t	maxTessellationEvaluationOutputComponents	tessellationShader
uint32_t	maxGeometryShaderInvocations	geometryShader

Type	Limit	Feature
uint32_t	maxGeometryInputComponents	geometryShader
uint32_t	maxGeometryOutputComponents	geometryShader
uint32_t	maxGeometryOutputVertices	geometryShader
uint32_t	maxGeometryTotalOutputComponents	geometryShader
uint32_t	maxFragmentInputComponents	-
uint32_t	maxFragmentOutputAttachments	-
uint32_t	maxFragmentDualSrcAttachments	dualSrcBlend
uint32_t	maxFragmentCombinedOutputResources	-
uint32_t	maxComputeSharedMemorySize	-
3 × uint32_t	maxComputeWorkGroupCount	-
uint32_t	maxComputeWorkGroupInvocations	-
3 × uint32_t	maxComputeWorkGroupSize	-
uint32_t	subPixelPrecisionBits	-
uint32_t	subTexelPrecisionBits	-
uint32_t	mipmapPrecisionBits	-
uint32_t	maxDrawIndexedIndexValue	fullDrawIndexUint32
uint32_t	maxDrawIndirectCount	multiDrawIndirect
float	maxSamplerLodBias	-
float	maxSamplerAnisotropy	samplerAnisotropy
uint32_t	maxViewports	multiViewport
2 × uint32_t	maxViewportDimensions	-
2 × float	viewportBoundsRange	-
uint32_t	viewportSubPixelBits	-
size_t	minMemoryMapAlignment	-
VkDeviceSize	minTexelBufferOffsetAlignment	-
VkDeviceSize	minUniformBufferOffsetAlignment	-
VkDeviceSize	minStorageBufferOffsetAlignment	-
int32_t	minTexelOffset	-
uint32_t	maxTexelOffset	-
int32_t	minTexelGatherOffset	shaderImageGatherExtended
uint32_t	maxTexelGatherOffset	shaderImageGatherExtended
float	minInterpolationOffset	sampleRateShading
float	maxInterpolationOffset	sampleRateShading
uint32_t	subPixelInterpolationOffsetBits	sampleRateShading
uint32_t	maxFramebufferWidth	-

Type	Limit	Feature
uint32_t	maxFramebufferHeight	-
uint32_t	maxFramebufferLayers	geometryShader, shaderOutputLayer
VkSampleCountFlags	framebufferColorSampleCounts	-
VkSampleCountFlags	framebufferIntegerColorSampleCounts	-
VkSampleCountFlags	framebufferDepthSampleCounts	-
VkSampleCountFlags	framebufferStencilSampleCounts	-
VkSampleCountFlags	framebufferNoAttachmentsSampleCounts	-
uint32_t	maxColorAttachments	-
VkSampleCountFlags	sampledImageColorSampleCounts	-
VkSampleCountFlags	sampledImageIntegerSampleCounts	-
VkSampleCountFlags	sampledImageDepthSampleCounts	-
VkSampleCountFlags	sampledImageStencilSampleCounts	-
VkSampleCountFlags	storageImageSampleCounts	shaderStorageImageMultisample
uint32_t	maxSampleMaskWords	-
VkBool32	timestampComputeAndGraphics	-
float	timestampPeriod	-
uint32_t	maxClipDistances	shaderClipDistance
uint32_t	maxCullDistances	shaderCullDistance
uint32_t	maxCombinedClipAndCullDistances	shaderCullDistance
uint32_t	discreteQueuePriorities	-
2 × float	pointSizeRange	largePoints
2 × float	lineWidthRange	wideLines
float	pointSizeGranularity	largePoints
float	lineWidthGranularity	wideLines
VkBool32	strictLines	-
VkBool32	standardSampleLocations	-

Type	Limit	Feature
VkDeviceSize	optimalBufferCopyOffsetAlignment	-
VkDeviceSize	optimalBufferCopyRowPitchAlignment	-
VkDeviceSize	nonCoherentAtomSize	-
uint32_t	maxDiscardRectangles	VK_EXT_discard_rectangles
VkBool32	filterMinmaxSingleComponentFormats	samplerFilterMinmax
VkBool32	filterMinmaxImageComponentMapping	samplerFilterMinmax
float	primitiveOverestimationSize	VK_EXT_conservative_rasterization
VkBool32	maxExtraPrimitiveOverestimationSize	VK_EXT_conservative_rasterization
float	extraPrimitiveOverestimationSizeGranularity	VK_EXT_conservative_rasterization
VkBool32	degenerateTriangleRasterized	VK_EXT_conservative_rasterization
float	degenerateLinesRasterized	VK_EXT_conservative_rasterization
VkBool32	fullyCoveredFragmentShaderInputVariable	VK_EXT_conservative_rasterization
VkBool32	conservativeRasterizationPostDepthCoverage	VK_EXT_conservative_rasterization
uint32_t	maxUpdateAfterBindDescriptorsInAllPools	descriptorIndexing
VkBool32	shaderUniformBufferArrayNonUniformIndexingNative	-
VkBool32	shaderSampledImageArrayNonUniformIndexingNative	-
VkBool32	shaderStorageBufferArrayNonUniformIndexingNative	-
VkBool32	shaderStorageImageArrayNonUniformIndexingNative	-
VkBool32	shaderInputAttachmentArrayNonUniformIndexingNative	-
uint32_t	maxPerStageDescriptorUpdateAfterBindSamplers	descriptorIndexing
uint32_t	maxPerStageDescriptorUpdateAfterBindUniformBuffers	descriptorIndexing
uint32_t	maxPerStageDescriptorUpdateAfterBindStorageBuffers	descriptorIndexing
uint32_t	maxPerStageDescriptorUpdateAfterBindSampledImages	descriptorIndexing
uint32_t	maxPerStageDescriptorUpdateAfterBindStorageImages	descriptorIndexing
uint32_t	maxPerStageDescriptorUpdateAfterBindInputAttachments	descriptorIndexing
uint32_t	maxPerStageUpdateAfterBindResources	descriptorIndexing

Type	Limit	Feature
uint32_t	maxDescriptorSetUpdateAfterBindSamplers	descriptorIndexing
uint32_t	maxDescriptorSetUpdateAfterBindUniformBuffers	descriptorIndexing
uint32_t	maxDescriptorSetUpdateAfterBindUniformBuffersDynamic	descriptorIndexing
uint32_t	maxDescriptorSetUpdateAfterBindStorageBuffers	descriptorIndexing
uint32_t	maxDescriptorSetUpdateAfterBindStorageBuffersDynamic	descriptorIndexing
uint32_t	maxDescriptorSetUpdateAfterBindSampledImages	descriptorIndexing
uint32_t	maxDescriptorSetUpdateAfterBindStorageImages	descriptorIndexing
uint32_t	maxDescriptorSetUpdateAfterBindInputAttachments	descriptorIndexing
uint32_t	maxVertexAttribDivisor	Vulkan 1.4, VK_KHR_vertex_attribute_divisor , VK_EXT_vertex_attribute_divisor
uint64_t	maxTimelineSemaphoreValueDifference	timelineSemaphore
uint32_t	lineSubPixelPrecisionBits	Vulkan 1.4, VK_KHR_line_rasterization , VK_EXT_line_rasterization
uint32_t	maxCustomBorderColorSamplers	VK_EXT_custom_border_color
VkDeviceSize	robustStorageBufferAccessSizeAlignment	VK_EXT_robustness2 , VK_KHR_robustness2
VkDeviceSize	robustUniformBufferAccessSizeAlignment	VK_EXT_robustness2 , VK_KHR_robustness2
2 × uint32_t	minFragmentShadingRateAttachmentTexelSize	attachmentFragmentShadingRate
2 × uint32_t	maxFragmentShadingRateAttachmentTexelSize	attachmentFragmentShadingRate
uint32_t	maxFragmentShadingRateAttachmentTexelSizeAspectRatio	attachmentFragmentShadingRate
VkBool32	primitiveFragmentShadingRateWithMultipleViewports	primitiveFragmentShadingRate
VkBool32	layeredShadingRateAttachments	attachmentFragmentShadingRate
VkBool32	fragmentShadingRateNonTrivialCombinerOps	pipelineFragmentShadingRate
2 × uint32_t	maxFragmentSize	pipelineFragmentShadingRate
uint32_t	maxFragmentSizeAspectRatio	pipelineFragmentShadingRate
uint32_t	maxFragmentShadingRateCoverageSamples	pipelineFragmentShadingRate
VkSampleCountFlagsBits	maxFragmentShadingRateRasterizationSamples	pipelineFragmentShadingRate

Type	Limit	Feature
VkBool32	fragmentShadingRateWithShaderDepthStencilWrites	pipelineFragmentShadingRate
VkBool32	fragmentShadingRateWithSampleMask	pipelineFragmentShadingRate
VkBool32	fragmentShadingRateWithShaderSampleMask	pipelineFragmentShadingRate
VkBool32	fragmentShadingRateWithConservativeRasterization	pipelineFragmentShadingRate
VkBool32	fragmentShadingRateWithFragmentShaderInterlock	pipelineFragmentShadingRate
VkBool32	fragmentShadingRateWithCustomSampleLocations	pipelineFragmentShadingRate
VkBool32	fragmentShadingRateStrictMultiplyCombiner	pipelineFragmentShadingRate
VkBool32	deviceNoDynamicHostAllocations	-
VkBool32	deviceDestroyFreesMemory	-
VkBool32	commandPoolMultipleCommandBuffersRecording	-
VkBool32	commandPoolResetCommandBuffer	-
VkBool32	commandBufferSimultaneousUse	-
VkBool32	secondaryCommandBufferNullOrImagelessFramebuffer	-
VkBool32	recycleDescriptorSetMemory	-
VkBool32	recyclePipelineMemory	-
uint32_t	maxRenderPassSubpasses	-
uint32_t	maxRenderPassDependencies	-
uint32_t	maxSubpassInputAttachments	-
uint32_t	maxSubpassPreserveAttachments	-
uint32_t	maxFramebufferAttachments	-
uint32_t	maxDescriptorSetLayoutBindings	-
uint32_t	maxQueryFaultCount	-
uint32_t	maxCallbackFaultCount	-
uint32_t	maxCommandPoolCommandBuffers	-
VkDeviceSize	maxCommandBufferSize	-
VkBool32	shaderSignedZeroInfNanPreserveFloat16	shaderFloat16

Table 49. Required Limits

Limit	Unsupported Limit	Supported Limit	Limit Type ¹
maxImageDimension1D	-	4096 (Vulkan Core) 8192 (Vulkan Roadmap 2022, Vulkan 1.4)	min

Limit	Unsupport ed Limit	Supported Limit	Limit Type ¹
maxImageDimension2D	-	4096 (Vulkan Core) 8192 (Vulkan Roadmap 2022, Vulkan 1.4)	min
maxImageDimension3D	-	256 (Vulkan Core) 512 (Vulkan 1.4)	min
maxImageDimensionCube	-	4096 (Vulkan Core) 8192 (Vulkan Roadmap 2022, Vulkan 1.4)	min
maxImageArrayLayers	-	256 (Vulkan Core) 2048 (Vulkan Roadmap 2022, Vulkan 1.4)	min
maxTexelBufferElements	-	65536	min
maxUniformBufferRange	-	16384 (Vulkan Core) 65536 (Vulkan Roadmap 2022, Vulkan 1.4)	min
maxStorageBufferRange	-	2 ²⁷	min
maxPushConstantsSize	-	128 (Vulkan Core) 256 (Vulkan 1.4)	min
maxMemoryAllocationCount	-	4096	min
maxSamplerAllocationCount	-	4000	min
bufferImageGranularity	-	131072 (Vulkan Core) 4096 (Vulkan Roadmap 2022, Vulkan 1.4)	max
sparseAddressSpaceSize	0	2 ³¹	min
maxBoundDescriptorSets	-	4 (Vulkan Core) 7 (Vulkan Roadmap 2024, Vulkan 1.4)	min
maxPerStageDescriptorSamplers	-	16	min

Limit	Unsupport ed Limit	Supported Limit	Limit Type ¹
maxPerStageDescriptorUniformBuffers	-	12 (Vulkan Core) 15 (Vulkan Roadmap 2022, Vulkan 1.4)	min
maxPerStageDescriptorStorageBuffers	-	4 (Vulkan Core) 30 (Vulkan Roadmap 2022)	min
maxPerStageDescriptorSampledImages	-	16 (Vulkan Core) 200 (Vulkan Roadmap 2022)	min
maxPerStageDescriptorStorageImages	-	4 (Vulkan Core) 144 (Vulkan Roadmap 2022)	min
maxPerStageDescriptorInputAttachments	-	4	min
maxPerStageResources	-	128 ² (Vulkan Core) 200 (Vulkan Roadmap 2022, Vulkan 1.4)	min
maxDescriptorSetSamplers	-	96 ⁸ (Vulkan Core) 576 (Vulkan Roadmap 2022)	min, $n \times$ PerStage
maxDescriptorSetUniformBuffers	-	72 ⁸ (Vulkan Core) 90 (Vulkan Roadmap 2022, Vulkan 1.4)	min, $n \times$ PerStage
maxDescriptorSetUniformBuffersDynamic	-	8	min
maxDescriptorSetStorageBuffers	-	24 ⁸ (Vulkan Core) 96 (Vulkan Roadmap 2022, Vulkan 1.4)	min, $n \times$ PerStage
maxDescriptorSetStorageBuffersDynamic	-	4	min
maxDescriptorSetSampledImages	-	96 ⁸ (Vulkan Core) 1800 (Vulkan Roadmap 2022)	min, $n \times$ PerStage
maxDescriptorSetStorageImages	-	24 ⁸ (Vulkan Core) 144 (Vulkan Roadmap 2022, Vulkan 1.4)	min, $n \times$ PerStage
maxDescriptorSetInputAttachments	-	4	min

Limit	Unsupport ed Limit	Supported Limit	Limit Type ¹
maxVertexInputAttributes	-	16	min
maxVertexInputBindings	-	16	min
maxVertexInputAttributeOffset	-	2047	min
maxVertexInputBindingStride	-	2048	min
maxVertexOutputComponents	-	64	min
maxTessellationGenerationLevel	0	64	min
maxTessellationPatchSize	0	32	min
maxTessellationControlPerVertexInputComponents	0	64	min
maxTessellationControlPerVertexOutputComponents	0	64	min
maxTessellationControlPerPatchOutputComponents	0	120	min
maxTessellationControlTotalOutputComponents	0	2048	min
maxTessellationEvaluationInputComponents	0	64	min
maxTessellationEvaluationOutputComponents	0	64	min
maxGeometryShaderInvocations	0	32	min
maxGeometryInputComponents	0	64	min
maxGeometryOutputComponents	0	64	min
maxGeometryOutputVertices	0	256	min
maxGeometryTotalOutputComponents	0	1024	min
maxFragmentInputComponents	-	64	min
maxFragmentOutputAttachments	-	4	min
maxFragmentDualSrcAttachments	0	1	min
maxFragmentCombinedOutputResources	-	4 (Vulkan Core) 16 (Vulkan Roadmap 2022, Vulkan 1.4)	min
maxComputeSharedMemorySize	-	16384	min
maxComputeWorkGroupCount	-	(65535,65535,65535)	min
maxComputeWorkGroupInvocations	-	128 (Vulkan Core) 256 (Vulkan Roadmap 2022, Vulkan 1.4)	min

Limit	Unsupport ed Limit	Supported Limit	Limit Type ¹
maxComputeWorkGroupSize	-	(128,128,64) (Vulkan Core) (256,256,64) (Vulkan Roadmap 2022, Vulkan 1.4)	min
subgroupSize	-	1/4 (Vulkan Core) 4 (Vulkan Roadmap 2022)	min
subgroupSupportedStages	-	VK_SHADER_STAGE_COMPUTE_BIT (Vulkan Core) VK_SHADER_STAGE_COMPUTE_BIT VK_SHADER_STAGE_FRAGMENT_BIT (Vulkan Roadmap 2022)	bitfield
subgroupSupportedOperations	-	VK_SUBGROUP_FEATURE_BASIC_BIT (Vulkan Core) VK_SUBGROUP_FEATURE_BASIC_BIT VK_SUBGROUP_FEATURE_VOTE_BIT VK_SUBGROUP_FEATURE_ARITHMETIC_BIT VK_SUBGROUP_FEATURE_BALLOT_BIT VK_SUBGROUP_FEATURE_SHUFFLE_BIT VK_SUBGROUP_FEATURE_SHUFFLE_RELATIVE_BIT VK_SUBGROUP_FEATURE_QUAD_BIT (Vulkan Roadmap 2022)	bitfield
shaderSignedZeroInfNanPreserveFloat16	-	- (Vulkan Core) VK_TRUE (Vulkan Roadmap 2022, Vulkan 1.4)	Boolean
shaderSignedZeroInfNanPreserveFloat32	-	- (Vulkan Core) VK_TRUE (Vulkan Roadmap 2022, Vulkan 1.4)	Boolean

Limit	Unsupport ed Limit	Supported Limit	Limit Type ¹
shaderRoundingModeRTEFloat16	-	VK_FALSE (Vulkan Core) VK_TRUE (Vulkan Roadmap 2024)	Boolean
shaderRoundingModeRTEFloat32	-	VK_FALSE (Vulkan Core) VK_TRUE (Vulkan Roadmap 2024)	Boolean
maxSubgroupSize	-	- (Vulkan Core) 4 (Vulkan Roadmap 2022)	min
subPixelPrecisionBits	-	4	min
subTexelPrecisionBits	-	4 (Vulkan Core) 8 (Vulkan Roadmap 2022, Vulkan 1.4)	min
mipmapPrecisionBits	-	4 (Vulkan Core) 6 (Vulkan Roadmap 2022, Vulkan 1.4)	min
maxDrawIndexedIndexValue	$2^{24}-1$	$2^{32}-1$	min
maxDrawIndirectCount	1	$2^{16}-1$	min
maxSamplerLodBias	-	2 (Vulkan Core) 14 (Vulkan Roadmap 2022, Vulkan 1.4)	min
maxSamplerAnisotropy	1	16	min
maxViewports	1	16	min
maxViewportDimensions ³	-	(4096,4096) (Vulkan Core) (7680,7680) (Vulkan 1.4)	min
viewportBoundsRange ⁴	-	(-8192,8191) (Vulkan Core) (-15360,15359) (Vulkan 1.4)	(max,min)
viewportSubPixelBits	-	0	min
minMemoryMapAlignment	-	64	min
minTexelBufferOffsetAlignment	-	256	max

Limit	Unsupport ed Limit	Supported Limit	Limit Type ¹
minUniformBufferOffsetAlignment	-	256	max
minStorageBufferOffsetAlignment	-	256	max
minTexelOffset	-	-8	max
maxTexelOffset	-	7	min
minTexelGatherOffset	0	-8	max
maxTexelGatherOffset	0	7	min
minInterpolationOffset	0.0	-0.5 ⁵	max
maxInterpolationOffset	0.0	0.5 - (1 ULP) ⁵	min
subPixelInterpolationOffsetBits	0	4 ⁵	min
maxFramebufferWidth	-	4096 (Vulkan Core) 7680 (Vulkan 1.4)	min
maxFramebufferHeight	-	4096 (Vulkan Core) 7680 (Vulkan 1.4)	min
maxFramebufferLayers	1	256	min
framebufferColorSampleCounts	-	(VK_SAMPLE_COUNT_1_BIT VK_SAMPLE_COUNT_4_BIT)	min
framebufferIntegerColorSampleCounts	-	(VK_SAMPLE_COUNT_1_BIT)	min
framebufferDepthSampleCounts	-	(VK_SAMPLE_COUNT_1_BIT VK_SAMPLE_COUNT_4_BIT)	min
framebufferStencilSampleCounts	-	(VK_SAMPLE_COUNT_1_BIT VK_SAMPLE_COUNT_4_BIT)	min
framebufferNoAttachmentsSampleCounts	-	(VK_SAMPLE_COUNT_1_BIT VK_SAMPLE_COUNT_4_BIT)	min
maxColorAttachments	-	4 (Vulkan Core) 7 (Vulkan Roadmap 2022) 8 (Vulkan Roadmap 2024, Vulkan 1.4)	min

Limit	Unsupport ed Limit	Supported Limit	Limit Type ¹
sampledImageColorSampleCounts	-	(VK_SAMPLE_COUNT_1_BIT VK_SAMPLE_COUNT_4_BIT)	min
sampledImageIntegerSampleCounts	-	VK_SAMPLE_COUNT_1_BIT	min
sampledImageDepthSampleCounts	-	(VK_SAMPLE_COUNT_1_BIT VK_SAMPLE_COUNT_4_BIT)	min
sampledImageStencilSampleCounts	-	(VK_SAMPLE_COUNT_1_BIT VK_SAMPLE_COUNT_4_BIT)	min
storageImageSampleCounts	VK_SAMPLE_COUNT_1_BIT	(VK_SAMPLE_COUNT_1_BIT VK_SAMPLE_COUNT_4_BIT)	min
maxSampleMaskWords	-	1	min
timestampComputeAndGraphics	-	- (Vulkan Core) VK_TRUE (Vulkan Roadmap 2024, Vulkan 1.4)	Boolean
timestampPeriod	-	-	duration
maxClipDistances	0	8	min
maxCullDistances	0	8	min
maxCombinedClipAndCullDistances	0	8	min
discreteQueuePriorities	-	2	min
pointSizeRange	(1.0,1.0)	(1.0,64.0 - ULP) ⁶ (Vulkan Core) (1.0,256.0 - pointSizeGranularity) (Vulkan 1.4)	(max,min)
lineWidthRange	(1.0,1.0)	(1.0,8.0 - ULP) ⁷	(max,min)
pointSizeGranularity	0.0	1.0 ⁶ (Vulkan Core) 0.125 (Vulkan Roadmap 2022, Vulkan 1.4)	max, fixed point increment

Limit	Unsupport ed Limit	Supported Limit	Limit Type ¹
lineWidthGranularity	0.0	1.0 ⁷ (Vulkan Core) 0.5 (Vulkan Roadmap 2022, Vulkan 1.4)	max, fixed point increment
strictLines	-	-	implementatio n-dependent
standardSampleLocations	-	- (Vulkan Core) VK_TRUE (Vulkan Roadmap 2022, Vulkan 1.4)	Boolean
optimalBufferCopyOffsetAlignment	-	-	recommendati on
optimalBufferCopyRowPitchAlignment	-	-	recommendati on
nonCoherentAtomSize	-	256	max
maxMultiviewViewCount	-	6	min
maxMultiviewInstanceIndex	-	2 ²⁷ -1	min
maxDiscardRectangles	0	4	min
sampleLocationSampleCounts	-	VK_SAMPLE_COUNT_4_ BIT	min
maxSampleLocationGridSize	-	(1,1)	min
sampleLocationCoordinateRange	-	(0.0, 0.9375)	(max,min)
sampleLocationSubPixelBits	-	4	min
variableSampleLocations	-	VK_FALSE	implementatio n-dependent
minImportedHostPointerAlignment	-	65536	max
filterMinmaxSingleComponentFormats	-	-	implementatio n-dependent
filterMinmaxImageComponentMapping	-	-	implementatio n-dependent
advancedBlendMaxColorAttachments	-	1	min
advancedBlendIndependentBlend	-	VK_FALSE	implementatio n-dependent
advancedBlendNonPremultipliedSrcColor	-	VK_FALSE	implementatio n-dependent
advancedBlendNonPremultipliedDstColor	-	VK_FALSE	implementatio n-dependent

Limit	Unsupport ed Limit	Supported Limit	Limit Type ¹
advancedBlendCorrelatedOverlap	-	VK_FALSE	implementatio n-dependent
advancedBlendAllOperations	-	VK_FALSE	implementatio n-dependent
maxPerSetDescriptors	-	1024	min
maxMemoryAllocationSize	-	2 ³⁰	min
primitiveOverestimationSize	-	0.0	min
maxExtraPrimitiveOverestimationSize	-	0.0	min
extraPrimitiveOverestimationSizeGranularity	-	0.0	min
primitiveUnderestimation	-	VK_FALSE	implementatio n-dependent
conservativePointAndLineRasterization	-	VK_FALSE	implementatio n-dependent
degenerateTrianglesRasterized	-	VK_FALSE	implementatio n-dependent
degenerateLinesRasterized	-	VK_FALSE	implementatio n-dependent
fullyCoveredFragmentShaderInputVariable	-	VK_FALSE	implementatio n-dependent
conservativeRasterizationPostDepthCoverage	-	VK_FALSE	implementatio n-dependent
maxUpdateAfterBindDescriptorsInAllPools	0	500000	min
shaderUniformBufferArrayNonUniformIndexingNative	-	VK_FALSE	implementatio n-dependent
shaderSampledImageArrayNonUniformIndexingNative	-	VK_FALSE	implementatio n-dependent
shaderStorageBufferArrayNonUniformIndexingNative	-	VK_FALSE	implementatio n-dependent
shaderStorageImageArrayNonUniformIndexingNative	-	VK_FALSE	implementatio n-dependent
shaderInputAttachmentArrayNonUniformIndexingNative	-	VK_FALSE	implementatio n-dependent
maxPerStageDescriptorUpdateAfterBindSamplers	0 ⁹	500000 ⁹	min
maxPerStageDescriptorUpdateAfterBindUniformBuffers	0 ⁹	12 ⁹	min
maxPerStageDescriptorUpdateAfterBindStorageBuffers	0 ⁹	500000 ⁹	min

Limit	Unsupport ed Limit	Supported Limit	Limit Type ¹
maxPerStageDescriptorUpdateAfterBindSampledImages	0 ⁹	500000 ⁹	min
maxPerStageDescriptorUpdateAfterBindStorageImages	0 ⁹	500000 ⁹	min
maxPerStageDescriptorUpdateAfterBindInputAttachments	0 ⁹	4 ⁹ (Vulkan Core) 7 (Vulkan Roadmap 2022)	min
maxPerStageUpdateAfterBindResources	0 ⁹	500000 ⁹	min
maxDescriptorSetUpdateAfterBindSamplers	0 ⁹	500000 ⁹	min
maxDescriptorSetUpdateAfterBindUniformBuffers	0 ⁹	72 ^{8 9}	min, $n \times$ PerStage
maxDescriptorSetUpdateAfterBindUniformBuffersDynamic	0 ⁹	8 ⁹	min
maxDescriptorSetUpdateAfterBindStorageBuffers	0 ⁹	500000 ⁹	min
maxDescriptorSetUpdateAfterBindStorageBuffersDynamic	0 ⁹	4 ⁹	min
maxDescriptorSetUpdateAfterBindSampledImages	0 ⁹	500000 ⁹	min
maxDescriptorSetUpdateAfterBindStorageImages	0 ⁹	500000 ⁹	min
maxDescriptorSetUpdateAfterBindInputAttachments	0 ⁹	4 ⁹	min
maxVertexAttribDivisor	-	2 ¹⁶ -1	min
maxTimelineSemaphoreValueDifference	-	2 ³¹ -1	min
lineSubPixelPrecisionBits	-	4	min
maxCustomBorderColorSamplers	-	32	min
robustStorageBufferAccessSizeAlignment	-	4	max
robustUniformBufferAccessSizeAlignment	-	256	max
minFragmentShadingRateAttachmentTexelSize	(0,0)	(32,32)	max
maxFragmentShadingRateAttachmentTexelSize	(0,0)	(8,8)	min
maxFragmentShadingRateAttachmentTexelSizeAspectRatio	0	1	min
primitiveFragmentShadingRateWithMultipleViewports	VK_FALSE	VK_FALSE	implementation-dependent
layeredShadingRateAttachments	VK_FALSE	VK_FALSE	implementation-dependent
fragmentShadingRateNonTrivialCombinerOps	-	VK_FALSE	implementation-dependent
maxFragmentSize	-	(2,2)	min
maxFragmentSizeAspectRatio	-	2	min

Limit	Unsupport ed Limit	Supported Limit	Limit Type ¹
maxFragmentShadingRateCoverageSamples	-	16	min
maxFragmentShadingRateRasterizationSamples	-	VK_SAMPLE_COUNT_4_BIT	min
fragmentShadingRateWithShaderDepthStencilWrites	-	VK_FALSE	implementatio n-dependent
fragmentShadingRateWithSampleMask	-	VK_FALSE	implementatio n-dependent
fragmentShadingRateWithShaderSampleMask	-	VK_FALSE	implementatio n-dependent
fragmentShadingRateWithConservativeRasterization	-	VK_FALSE	implementatio n-dependent
fragmentShadingRateWithFragmentShaderInterlock	-	VK_FALSE	implementatio n-dependent
fragmentShadingRateWithCustomSampleLocations	-	VK_FALSE	implementatio n-dependent
fragmentShadingRateStrictMultiplyCombiner	-	VK_FALSE	implementatio n-dependent
deviceNoDynamicHostAllocations	-	-	implementatio n-dependent
deviceDestroyFreesMemory	-	-	implementatio n-dependent
commandPoolMultipleCommandBuffersRecording	-	-	implementatio n-dependent
commandPoolResetCommandBuffer	-	-	implementatio n-dependent
commandBufferSimultaneousUse	-	-	implementatio n-dependent
secondaryCommandBufferNullOrImagelessFramebuffer	-	-	implementatio n-dependent
recycleDescriptorSetMemory	-	-	implementatio n-dependent
recyclePipelineMemory	-	-	implementatio n-dependent
maxRenderPassSubpasses	-	1	min
maxRenderPassDependencies	-	18	min
maxSubpassInputAttachments	-	0	min
maxSubpassPreserveAttachments	-	0	min

Limit	Unsupport ed Limit	Supported Limit	Limit Type ¹
<code>maxFramebufferAttachments</code>	-	9^{11}	min
<code>maxDescriptorSetLayoutBindings</code>	-	64	min
<code>maxQueryFaultCount</code>	-	16	min
<code>maxCallbackFaultCount</code>	-	1	min
<code>maxCommandPoolCommandBuffers</code>	-	256	min
<code>maxCommandBufferSize</code>	-	2^{20}	min

1

The **Limit Type** column specifies the limit is either the minimum limit all implementations **must** support, the maximum limit all implementations **must** support, or the exact value all implementations **must** support. For bitmasks a minimum limit is the least bits all implementations **must** set, but they **may** have additional bits set beyond this minimum.

2

The `maxPerStageResources` **must** be at least the smallest of the following:

- the sum of the `maxPerStageDescriptorUniformBuffers`, `maxPerStageDescriptorStorageBuffers`, `maxPerStageDescriptorSampledImages`, `maxPerStageDescriptorStorageImages`, `maxPerStageDescriptorInputAttachments`, `maxColorAttachments` limits, or
- 128.

It **may** not be possible to reach this limit in every stage.

3

See `maxViewportDimensions` for the **required** relationship to other limits.

4

See `viewportBoundsRange` for the **required** relationship to other limits.

5

The values `minInterpolationOffset` and `maxInterpolationOffset` describe the closed interval of supported interpolation offsets: `[minInterpolationOffset, maxInterpolationOffset]`. The ULP is determined by `subPixelInterpolationOffsetBits`. If `subPixelInterpolationOffsetBits` is 4, this provides increments of $(1/2^4) = 0.0625$, and thus the range of supported interpolation offsets would be `[-0.5, 0.4375]`.

6

The point size ULP is determined by `pointSizeGranularity`. If the `pointSizeGranularity` is 0.125, the range of supported point sizes **must** be at least `[1.0, 63.875]`.

7

The line width ULP is determined by `lineWidthGranularity`. If the `lineWidthGranularity` is 0.0625, the range of supported line widths **must** be at least `[1.0, 7.9375]`.

8

The minimum `maxDescriptorSet*` limit is n times the corresponding *specification* minimum `maxPerStageDescriptor*` limit, where n is the number of shader stages supported by the `VkPhysicalDevice`. If all shader stages are supported, $n = 6$ (vertex, tessellation control, tessellation evaluation, geometry, fragment, compute).

9

The `UpdateAfterBind` descriptor limits **must** each be greater than or equal to the corresponding `non-UpdateAfterBind` limit.

11

`maxFramebufferAttachments` **must** be greater than or equal to two times `maxColorAttachments` (for color and resolve attachments) plus one (for the depth/stencil attachment), or else **must** be equal to $2^{32}-1$.

34.2. Additional Multisampling Capabilities

To query additional multisampling capabilities which **may** be supported for a specific sample count, beyond the minimum capabilities described for [Limits](#) above, call:

```
// Provided by VK_EXT_sample_locations
void vkGetPhysicalDeviceMultisamplePropertiesEXT(
    VkPhysicalDevice          physicalDevice,
    VkSampleCountFlagBits    samples,
    VkMultisamplePropertiesEXT* pMultisampleProperties);
```

- `physicalDevice` is the physical device from which to query the additional multisampling capabilities.
- `samples` is a `VkSampleCountFlagBits` value specifying the sample count to query capabilities for.
- `pMultisampleProperties` is a pointer to a `VkMultisamplePropertiesEXT` structure in which information about additional multisampling capabilities specific to the sample count is returned.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceMultisamplePropertiesEXT-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceMultisamplePropertiesEXT-samples-parameter `samples` **must** be a valid `VkSampleCountFlagBits` value
- VUID-vkGetPhysicalDeviceMultisamplePropertiesEXT-pMultisampleProperties-parameter `pMultisampleProperties` **must** be a valid pointer to a `VkMultisamplePropertiesEXT` structure

The `VkMultisamplePropertiesEXT` structure is defined as

```
// Provided by VK_EXT_sample_locations
typedef struct VkMultisamplePropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkExtent2D         maxSampleLocationGridSize;
} VkMultisamplePropertiesEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `maxSampleLocationGridSize` is the maximum size of the pixel grid in which sample locations **can** vary.

Valid Usage (Implicit)

- VUID-VkMultisamplePropertiesEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_MULTISAMPLE_PROPERTIES_EXT`
- VUID-VkMultisamplePropertiesEXT-pNext-pNext
`pNext` **must** be `NULL`

If the sample count for which additional multisampling capabilities are requested using `vkGetPhysicalDeviceMultisamplePropertiesEXT` is set in `sampleLocationSampleCounts` the `width` and `height` members of `VkMultisamplePropertiesEXT::maxSampleLocationGridSize` **must** be greater than or equal to the corresponding members of `maxSampleLocationGridSize`, respectively, otherwise both members **must** be `0`.

Chapter 35. Formats

Supported buffer and image formats **may** vary across implementations. A minimum set of format features are guaranteed, but others **must** be explicitly queried before use to ensure they are supported by the implementation.

The features for the set of formats ([VkFormat](#)) supported by the implementation are queried individually using the [vkGetPhysicalDeviceFormatProperties](#) command.

35.1. Format Definition

The following image formats **can** be passed to, and **may** be returned from Vulkan commands. The memory required to store each format is discussed with that format, and also summarized in the [Representation and Texel Block Size](#) section and the [Compatible formats](#) table.

```
// Provided by VK_VERSION_1_0
typedef enum VkFormat {
    VK_FORMAT_UNDEFINED = 0,
    VK_FORMAT_R4G4_UNORM_PACK8 = 1,
    VK_FORMAT_R4G4B4A4_UNORM_PACK16 = 2,
    VK_FORMAT_B4G4R4A4_UNORM_PACK16 = 3,
    VK_FORMAT_R5G6B5_UNORM_PACK16 = 4,
    VK_FORMAT_B5G6R5_UNORM_PACK16 = 5,
    VK_FORMAT_R5G5B5A1_UNORM_PACK16 = 6,
    VK_FORMAT_B5G5R5A1_UNORM_PACK16 = 7,
    VK_FORMAT_A1R5G5B5_UNORM_PACK16 = 8,
    VK_FORMAT_R8_UNORM = 9,
    VK_FORMAT_R8_SNORM = 10,
    VK_FORMAT_R8_USCALED = 11,
    VK_FORMAT_R8_SSCALED = 12,
    VK_FORMAT_R8_UINT = 13,
    VK_FORMAT_R8_SINT = 14,
    VK_FORMAT_R8_SRGB = 15,
    VK_FORMAT_R8G8_UNORM = 16,
    VK_FORMAT_R8G8_SNORM = 17,
    VK_FORMAT_R8G8_USCALED = 18,
    VK_FORMAT_R8G8_SSCALED = 19,
    VK_FORMAT_R8G8_UINT = 20,
    VK_FORMAT_R8G8_SINT = 21,
    VK_FORMAT_R8G8_SRGB = 22,
    VK_FORMAT_R8G8B8_UNORM = 23,
    VK_FORMAT_R8G8B8_SNORM = 24,
    VK_FORMAT_R8G8B8_USCALED = 25,
    VK_FORMAT_R8G8B8_SSCALED = 26,
    VK_FORMAT_R8G8B8_UINT = 27,
    VK_FORMAT_R8G8B8_SINT = 28,
    VK_FORMAT_R8G8B8_SRGB = 29,
    VK_FORMAT_B8G8R8_UNORM = 30,
    VK_FORMAT_B8G8R8_SNORM = 31,
```

```
VK_FORMAT_B8G8R8_USCALED = 32,  
VK_FORMAT_B8G8R8_SSCALED = 33,  
VK_FORMAT_B8G8R8_UINT = 34,  
VK_FORMAT_B8G8R8_SINT = 35,  
VK_FORMAT_B8G8R8_SRGB = 36,  
VK_FORMAT_R8G8B8A8_UNORM = 37,  
VK_FORMAT_R8G8B8A8_SNORM = 38,  
VK_FORMAT_R8G8B8A8_USCALED = 39,  
VK_FORMAT_R8G8B8A8_SSCALED = 40,  
VK_FORMAT_R8G8B8A8_UINT = 41,  
VK_FORMAT_R8G8B8A8_SINT = 42,  
VK_FORMAT_R8G8B8A8_SRGB = 43,  
VK_FORMAT_B8G8R8A8_UNORM = 44,  
VK_FORMAT_B8G8R8A8_SNORM = 45,  
VK_FORMAT_B8G8R8A8_USCALED = 46,  
VK_FORMAT_B8G8R8A8_SSCALED = 47,  
VK_FORMAT_B8G8R8A8_UINT = 48,  
VK_FORMAT_B8G8R8A8_SINT = 49,  
VK_FORMAT_B8G8R8A8_SRGB = 50,  
VK_FORMAT_A8B8G8R8_UNORM_PACK32 = 51,  
VK_FORMAT_A8B8G8R8_SNORM_PACK32 = 52,  
VK_FORMAT_A8B8G8R8_USCALED_PACK32 = 53,  
VK_FORMAT_A8B8G8R8_SSCALED_PACK32 = 54,  
VK_FORMAT_A8B8G8R8_UINT_PACK32 = 55,  
VK_FORMAT_A8B8G8R8_SINT_PACK32 = 56,  
VK_FORMAT_A8B8G8R8_SRGB_PACK32 = 57,  
VK_FORMAT_A2R10G10B10_UNORM_PACK32 = 58,  
VK_FORMAT_A2R10G10B10_SNORM_PACK32 = 59,  
VK_FORMAT_A2R10G10B10_USCALED_PACK32 = 60,  
VK_FORMAT_A2R10G10B10_SSCALED_PACK32 = 61,  
VK_FORMAT_A2R10G10B10_UINT_PACK32 = 62,  
VK_FORMAT_A2R10G10B10_SINT_PACK32 = 63,  
VK_FORMAT_A2B10G10R10_UNORM_PACK32 = 64,  
VK_FORMAT_A2B10G10R10_SNORM_PACK32 = 65,  
VK_FORMAT_A2B10G10R10_USCALED_PACK32 = 66,  
VK_FORMAT_A2B10G10R10_SSCALED_PACK32 = 67,  
VK_FORMAT_A2B10G10R10_UINT_PACK32 = 68,  
VK_FORMAT_A2B10G10R10_SINT_PACK32 = 69,  
VK_FORMAT_R16_UNORM = 70,  
VK_FORMAT_R16_SNORM = 71,  
VK_FORMAT_R16_USCALED = 72,  
VK_FORMAT_R16_SSCALED = 73,  
VK_FORMAT_R16_UINT = 74,  
VK_FORMAT_R16_SINT = 75,  
VK_FORMAT_R16_SFLOAT = 76,  
VK_FORMAT_R16G16_UNORM = 77,  
VK_FORMAT_R16G16_SNORM = 78,  
VK_FORMAT_R16G16_USCALED = 79,  
VK_FORMAT_R16G16_SSCALED = 80,  
VK_FORMAT_R16G16_UINT = 81,  
VK_FORMAT_R16G16_SINT = 82,
```

```
VK_FORMAT_R16G16_SFLOAT = 83,  
VK_FORMAT_R16G16B16_UNORM = 84,  
VK_FORMAT_R16G16B16_SNORM = 85,  
VK_FORMAT_R16G16B16_USCALED = 86,  
VK_FORMAT_R16G16B16_SSCALED = 87,  
VK_FORMAT_R16G16B16_UINT = 88,  
VK_FORMAT_R16G16B16_SINT = 89,  
VK_FORMAT_R16G16B16_SFLOAT = 90,  
VK_FORMAT_R16G16B16A16_UNORM = 91,  
VK_FORMAT_R16G16B16A16_SNORM = 92,  
VK_FORMAT_R16G16B16A16_USCALED = 93,  
VK_FORMAT_R16G16B16A16_SSCALED = 94,  
VK_FORMAT_R16G16B16A16_UINT = 95,  
VK_FORMAT_R16G16B16A16_SINT = 96,  
VK_FORMAT_R16G16B16A16_SFLOAT = 97,  
VK_FORMAT_R32_UINT = 98,  
VK_FORMAT_R32_SINT = 99,  
VK_FORMAT_R32_SFLOAT = 100,  
VK_FORMAT_R32G32_UINT = 101,  
VK_FORMAT_R32G32_SINT = 102,  
VK_FORMAT_R32G32_SFLOAT = 103,  
VK_FORMAT_R32G32B32_UINT = 104,  
VK_FORMAT_R32G32B32_SINT = 105,  
VK_FORMAT_R32G32B32_SFLOAT = 106,  
VK_FORMAT_R32G32B32A32_UINT = 107,  
VK_FORMAT_R32G32B32A32_SINT = 108,  
VK_FORMAT_R32G32B32A32_SFLOAT = 109,  
VK_FORMAT_R64_UINT = 110,  
VK_FORMAT_R64_SINT = 111,  
VK_FORMAT_R64_SFLOAT = 112,  
VK_FORMAT_R64G64_UINT = 113,  
VK_FORMAT_R64G64_SINT = 114,  
VK_FORMAT_R64G64_SFLOAT = 115,  
VK_FORMAT_R64G64B64_UINT = 116,  
VK_FORMAT_R64G64B64_SINT = 117,  
VK_FORMAT_R64G64B64_SFLOAT = 118,  
VK_FORMAT_R64G64B64A64_UINT = 119,  
VK_FORMAT_R64G64B64A64_SINT = 120,  
VK_FORMAT_R64G64B64A64_SFLOAT = 121,  
VK_FORMAT_B10G11R11_UFLOAT_PACK32 = 122,  
VK_FORMAT_E5B9G9R9_UFLOAT_PACK32 = 123,  
VK_FORMAT_D16_UNORM = 124,  
VK_FORMAT_X8_D24_UNORM_PACK32 = 125,  
VK_FORMAT_D32_SFLOAT = 126,  
VK_FORMAT_S8_UINT = 127,  
VK_FORMAT_D16_UNORM_S8_UINT = 128,  
VK_FORMAT_D24_UNORM_S8_UINT = 129,  
VK_FORMAT_D32_SFLOAT_S8_UINT = 130,  
VK_FORMAT_BC1_RGB_UNORM_BLOCK = 131,  
VK_FORMAT_BC1_RGB_SRGB_BLOCK = 132,  
VK_FORMAT_BC1_RGBA_UNORM_BLOCK = 133,
```



```
VK_FORMAT_BC1_RGBA_SRGB_BLOCK = 134,  
VK_FORMAT_BC2_UNORM_BLOCK = 135,  
VK_FORMAT_BC2_SRGB_BLOCK = 136,  
VK_FORMAT_BC3_UNORM_BLOCK = 137,  
VK_FORMAT_BC3_SRGB_BLOCK = 138,  
VK_FORMAT_BC4_UNORM_BLOCK = 139,  
VK_FORMAT_BC4_SNORM_BLOCK = 140,  
VK_FORMAT_BC5_UNORM_BLOCK = 141,  
VK_FORMAT_BC5_SNORM_BLOCK = 142,  
VK_FORMAT_BC6H_UFLOAT_BLOCK = 143,  
VK_FORMAT_BC6H_SFLOAT_BLOCK = 144,  
VK_FORMAT_BC7_UNORM_BLOCK = 145,  
VK_FORMAT_BC7_SRGB_BLOCK = 146,  
VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK = 147,  
VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK = 148,  
VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK = 149,  
VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK = 150,  
VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK = 151,  
VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK = 152,  
VK_FORMAT_EAC_R11_UNORM_BLOCK = 153,  
VK_FORMAT_EAC_R11_SNORM_BLOCK = 154,  
VK_FORMAT_EAC_R11G11_UNORM_BLOCK = 155,  
VK_FORMAT_EAC_R11G11_SNORM_BLOCK = 156,  
VK_FORMAT_ASTC_4x4_UNORM_BLOCK = 157,  
VK_FORMAT_ASTC_4x4_SRGB_BLOCK = 158,  
VK_FORMAT_ASTC_5x4_UNORM_BLOCK = 159,  
VK_FORMAT_ASTC_5x4_SRGB_BLOCK = 160,  
VK_FORMAT_ASTC_5x5_UNORM_BLOCK = 161,  
VK_FORMAT_ASTC_5x5_SRGB_BLOCK = 162,  
VK_FORMAT_ASTC_6x5_UNORM_BLOCK = 163,  
VK_FORMAT_ASTC_6x5_SRGB_BLOCK = 164,  
VK_FORMAT_ASTC_6x6_UNORM_BLOCK = 165,  
VK_FORMAT_ASTC_6x6_SRGB_BLOCK = 166,  
VK_FORMAT_ASTC_8x5_UNORM_BLOCK = 167,  
VK_FORMAT_ASTC_8x5_SRGB_BLOCK = 168,  
VK_FORMAT_ASTC_8x6_UNORM_BLOCK = 169,  
VK_FORMAT_ASTC_8x6_SRGB_BLOCK = 170,  
VK_FORMAT_ASTC_8x8_UNORM_BLOCK = 171,  
VK_FORMAT_ASTC_8x8_SRGB_BLOCK = 172,  
VK_FORMAT_ASTC_10x5_UNORM_BLOCK = 173,  
VK_FORMAT_ASTC_10x5_SRGB_BLOCK = 174,  
VK_FORMAT_ASTC_10x6_UNORM_BLOCK = 175,  
VK_FORMAT_ASTC_10x6_SRGB_BLOCK = 176,  
VK_FORMAT_ASTC_10x8_UNORM_BLOCK = 177,  
VK_FORMAT_ASTC_10x8_SRGB_BLOCK = 178,  
VK_FORMAT_ASTC_10x10_UNORM_BLOCK = 179,  
VK_FORMAT_ASTC_10x10_SRGB_BLOCK = 180,  
VK_FORMAT_ASTC_12x10_UNORM_BLOCK = 181,  
VK_FORMAT_ASTC_12x10_SRGB_BLOCK = 182,  
VK_FORMAT_ASTC_12x12_UNORM_BLOCK = 183,  
VK_FORMAT_ASTC_12x12_SRGB_BLOCK = 184,
```

```
// Provided by VK_VERSION_1_1
VK_FORMAT_G8B8G8R8_422_UNORM = 1000156000,
// Provided by VK_VERSION_1_1
VK_FORMAT_B8G8R8G8_422_UNORM = 1000156001,
// Provided by VK_VERSION_1_1
VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM = 1000156002,
// Provided by VK_VERSION_1_1
VK_FORMAT_G8_B8R8_2PLANE_420_UNORM = 1000156003,
// Provided by VK_VERSION_1_1
VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM = 1000156004,
// Provided by VK_VERSION_1_1
VK_FORMAT_G8_B8R8_2PLANE_422_UNORM = 1000156005,
// Provided by VK_VERSION_1_1
VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM = 1000156006,
// Provided by VK_VERSION_1_1
VK_FORMAT_R10X6_UNORM_PACK16 = 1000156007,
// Provided by VK_VERSION_1_1
VK_FORMAT_R10X6G10X6_UNORM_2PACK16 = 1000156008,
// Provided by VK_VERSION_1_1
VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16 = 1000156009,
// Provided by VK_VERSION_1_1
VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16 = 1000156010,
// Provided by VK_VERSION_1_1
VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16 = 1000156011,
// Provided by VK_VERSION_1_1
VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16 = 1000156012,
// Provided by VK_VERSION_1_1
VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16 = 1000156013,
// Provided by VK_VERSION_1_1
VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16 = 1000156014,
// Provided by VK_VERSION_1_1
VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16 = 1000156015,
// Provided by VK_VERSION_1_1
VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16 = 1000156016,
// Provided by VK_VERSION_1_1
VK_FORMAT_R12X4_UNORM_PACK16 = 1000156017,
// Provided by VK_VERSION_1_1
VK_FORMAT_R12X4G12X4_UNORM_2PACK16 = 1000156018,
// Provided by VK_VERSION_1_1
VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16 = 1000156019,
// Provided by VK_VERSION_1_1
VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16 = 1000156020,
// Provided by VK_VERSION_1_1
VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16 = 1000156021,
// Provided by VK_VERSION_1_1
VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16 = 1000156022,
// Provided by VK_VERSION_1_1
VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16 = 1000156023,
// Provided by VK_VERSION_1_1
VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16 = 1000156024,
// Provided by VK_VERSION_1_1
```

```

VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16 = 1000156025,
// Provided by VK_VERSION_1_1
VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16 = 1000156026,
// Provided by VK_VERSION_1_1
VK_FORMAT_G16B16G16R16_422_UNORM = 1000156027,
// Provided by VK_VERSION_1_1
VK_FORMAT_B16G16R16G16_422_UNORM = 1000156028,
// Provided by VK_VERSION_1_1
VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM = 1000156029,
// Provided by VK_VERSION_1_1
VK_FORMAT_G16_B16R16_2PLANE_420_UNORM = 1000156030,
// Provided by VK_VERSION_1_1
VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM = 1000156031,
// Provided by VK_VERSION_1_1
VK_FORMAT_G16_B16R16_2PLANE_422_UNORM = 1000156032,
// Provided by VK_VERSION_1_1
VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM = 1000156033,
VK_FORMAT_G8_B8R8_2PLANE_444_UNORM = 1000330000,
VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16 = 1000330001,
VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16 = 1000330002,
VK_FORMAT_G16_B16R16_2PLANE_444_UNORM = 1000330003,
VK_FORMAT_A4R4G4B4_UNORM_PACK16 = 1000340000,
VK_FORMAT_A4B4G4R4_UNORM_PACK16 = 1000340001,
VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK = 1000066000,
VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK = 1000066001,
VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK = 1000066002,
VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK = 1000066003,
VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK = 1000066004,
VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK = 1000066005,
VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK = 1000066006,
VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK = 1000066007,
VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK = 1000066008,
VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK = 1000066009,
VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK = 1000066010,
VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK = 1000066011,
VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK = 1000066012,
VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK = 1000066013,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK,

```

```

// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK,
// Provided by VK_EXT_texture_compression_astc_hdr
VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK_EXT = VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK,
// Provided by VK_EXT_ycbcr_2plane_444_formats
VK_FORMAT_G8_B8R8_2PLANE_444_UNORM_EXT = VK_FORMAT_G8_B8R8_2PLANE_444_UNORM,
// Provided by VK_EXT_ycbcr_2plane_444_formats
VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16_EXT =
VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16,
// Provided by VK_EXT_ycbcr_2plane_444_formats
VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16_EXT =
VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16,
// Provided by VK_EXT_ycbcr_2plane_444_formats
VK_FORMAT_G16_B16R16_2PLANE_444_UNORM_EXT = VK_FORMAT_G16_B16R16_2PLANE_444_UNORM,
// Provided by VK_EXT_4444_formats
VK_FORMAT_A4R4G4B4_UNORM_PACK16_EXT = VK_FORMAT_A4R4G4B4_UNORM_PACK16,
// Provided by VK_EXT_4444_formats
VK_FORMAT_A4B4G4R4_UNORM_PACK16_EXT = VK_FORMAT_A4B4G4R4_UNORM_PACK16,
} VkFormat;

```

- **VK_FORMAT_UNDEFINED** specifies that the format is not specified.
- **VK_FORMAT_R4G4_UNORM_PACK8** specifies a two-component, 8-bit packed unsigned normalized format that has a 4-bit R component in bits 4..7, and a 4-bit G component in bits 0..3.
- **VK_FORMAT_R4G4B4A4_UNORM_PACK16** specifies a four-component, 16-bit packed unsigned normalized format that has a 4-bit R component in bits 12..15, a 4-bit G component in bits 8..11, a 4-bit B component in bits 4..7, and a 4-bit A component in bits 0..3.
- **VK_FORMAT_B4G4R4A4_UNORM_PACK16** specifies a four-component, 16-bit packed unsigned normalized format that has a 4-bit B component in bits 12..15, a 4-bit G component in bits 8..11, a 4-bit R component in bits 4..7, and a 4-bit A component in bits 0..3.
- **VK_FORMAT_A4R4G4B4_UNORM_PACK16** specifies a four-component, 16-bit packed unsigned normalized format that has a 4-bit A component in bits 12..15, a 4-bit R component in bits 8..11, a 4-bit G component in bits 4..7, and a 4-bit B component in bits 0..3.
- **VK_FORMAT_A4B4G4R4_UNORM_PACK16** specifies a four-component, 16-bit packed unsigned normalized format that has a 4-bit A component in bits 12..15, a 4-bit B component in bits 8..11, a 4-bit G component in bits 4..7, and a 4-bit R component in bits 0..3.
- **VK_FORMAT_R5G6B5_UNORM_PACK16** specifies a three-component, 16-bit packed unsigned normalized format that has a 5-bit R component in bits 11..15, a 6-bit G component in bits 5..10, and a 5-bit B

component in bits 0..4.

- **VK_FORMAT_B5G6R5_UNORM_PACK16** specifies a three-component, 16-bit packed unsigned normalized format that has a 5-bit B component in bits 11..15, a 6-bit G component in bits 5..10, and a 5-bit R component in bits 0..4.
- **VK_FORMAT_R5G5B5A1_UNORM_PACK16** specifies a four-component, 16-bit packed unsigned normalized format that has a 5-bit R component in bits 11..15, a 5-bit G component in bits 6..10, a 5-bit B component in bits 1..5, and a 1-bit A component in bit 0.
- **VK_FORMAT_B5G5R5A1_UNORM_PACK16** specifies a four-component, 16-bit packed unsigned normalized format that has a 5-bit B component in bits 11..15, a 5-bit G component in bits 6..10, a 5-bit R component in bits 1..5, and a 1-bit A component in bit 0.
- **VK_FORMAT_A1R5G5B5_UNORM_PACK16** specifies a four-component, 16-bit packed unsigned normalized format that has a 1-bit A component in bit 15, a 5-bit R component in bits 10..14, a 5-bit G component in bits 5..9, and a 5-bit B component in bits 0..4.
- **VK_FORMAT_R8_UNORM** specifies a one-component, 8-bit unsigned normalized format that has a single 8-bit R component.
- **VK_FORMAT_R8_SNORM** specifies a one-component, 8-bit signed normalized format that has a single 8-bit R component.
- **VK_FORMAT_R8_USCALED** specifies a one-component, 8-bit unsigned scaled integer format that has a single 8-bit R component.
- **VK_FORMAT_R8_SSCALED** specifies a one-component, 8-bit signed scaled integer format that has a single 8-bit R component.
- **VK_FORMAT_R8_UINT** specifies a one-component, 8-bit unsigned integer format that has a single 8-bit R component.
- **VK_FORMAT_R8_SINT** specifies a one-component, 8-bit signed integer format that has a single 8-bit R component.
- **VK_FORMAT_R8_SRGB** specifies a one-component, 8-bit unsigned normalized format that has a single 8-bit R component stored with sRGB nonlinear encoding.
- **VK_FORMAT_R8G8_UNORM** specifies a two-component, 16-bit unsigned normalized format that has an 8-bit R component in byte 0, and an 8-bit G component in byte 1.
- **VK_FORMAT_R8G8_SNORM** specifies a two-component, 16-bit signed normalized format that has an 8-bit R component in byte 0, and an 8-bit G component in byte 1.
- **VK_FORMAT_R8G8_USCALED** specifies a two-component, 16-bit unsigned scaled integer format that has an 8-bit R component in byte 0, and an 8-bit G component in byte 1.
- **VK_FORMAT_R8G8_SSCALED** specifies a two-component, 16-bit signed scaled integer format that has an 8-bit R component in byte 0, and an 8-bit G component in byte 1.
- **VK_FORMAT_R8G8_UINT** specifies a two-component, 16-bit unsigned integer format that has an 8-bit R component in byte 0, and an 8-bit G component in byte 1.
- **VK_FORMAT_R8G8_SINT** specifies a two-component, 16-bit signed integer format that has an 8-bit R component in byte 0, and an 8-bit G component in byte 1.
- **VK_FORMAT_R8G8_SRGB** specifies a two-component, 16-bit unsigned normalized format that has an 8-bit R component stored with sRGB nonlinear encoding in byte 0, and an 8-bit G component

stored with sRGB nonlinear encoding in byte 1.

- **VK_FORMAT_R8G8B8_UNORM** specifies a three-component, 24-bit unsigned normalized format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, and an 8-bit B component in byte 2.
- **VK_FORMAT_R8G8B8_SNORM** specifies a three-component, 24-bit signed normalized format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, and an 8-bit B component in byte 2.
- **VK_FORMAT_R8G8B8_USCALED** specifies a three-component, 24-bit unsigned scaled format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, and an 8-bit B component in byte 2.
- **VK_FORMAT_R8G8B8_SSCALED** specifies a three-component, 24-bit signed scaled format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, and an 8-bit B component in byte 2.
- **VK_FORMAT_R8G8B8_UINT** specifies a three-component, 24-bit unsigned integer format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, and an 8-bit B component in byte 2.
- **VK_FORMAT_R8G8B8_SINT** specifies a three-component, 24-bit signed integer format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, and an 8-bit B component in byte 2.
- **VK_FORMAT_R8G8B8_SRGB** specifies a three-component, 24-bit unsigned normalized format that has an 8-bit R component stored with sRGB nonlinear encoding in byte 0, an 8-bit G component stored with sRGB nonlinear encoding in byte 1, and an 8-bit B component stored with sRGB nonlinear encoding in byte 2.
- **VK_FORMAT_B8G8R8_UNORM** specifies a three-component, 24-bit unsigned normalized format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, and an 8-bit R component in byte 2.
- **VK_FORMAT_B8G8R8_SNORM** specifies a three-component, 24-bit signed normalized format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, and an 8-bit R component in byte 2.
- **VK_FORMAT_B8G8R8_USCALED** specifies a three-component, 24-bit unsigned scaled format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, and an 8-bit R component in byte 2.
- **VK_FORMAT_B8G8R8_SSCALED** specifies a three-component, 24-bit signed scaled format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, and an 8-bit R component in byte 2.
- **VK_FORMAT_B8G8R8_UINT** specifies a three-component, 24-bit unsigned integer format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, and an 8-bit R component in byte 2.
- **VK_FORMAT_B8G8R8_SINT** specifies a three-component, 24-bit signed integer format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, and an 8-bit R component in byte 2.
- **VK_FORMAT_B8G8R8_SRGB** specifies a three-component, 24-bit unsigned normalized format that has an 8-bit B component stored with sRGB nonlinear encoding in byte 0, an 8-bit G component stored with sRGB nonlinear encoding in byte 1, and an 8-bit R component stored with sRGB nonlinear encoding in byte 2.
- **VK_FORMAT_R8G8B8A8_UNORM** specifies a four-component, 32-bit unsigned normalized format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_R8G8B8A8_SNORM** specifies a four-component, 32-bit signed normalized format that has

an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.

- **VK_FORMAT_R8G8B8A8_USCALED** specifies a four-component, 32-bit unsigned scaled format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_R8G8B8A8_SSCALED** specifies a four-component, 32-bit signed scaled format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_R8G8B8A8_UINT** specifies a four-component, 32-bit unsigned integer format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_R8G8B8A8_SINT** specifies a four-component, 32-bit signed integer format that has an 8-bit R component in byte 0, an 8-bit G component in byte 1, an 8-bit B component in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_R8G8B8A8_SRGB** specifies a four-component, 32-bit unsigned normalized format that has an 8-bit R component stored with sRGB nonlinear encoding in byte 0, an 8-bit G component stored with sRGB nonlinear encoding in byte 1, an 8-bit B component stored with sRGB nonlinear encoding in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_B8G8R8A8_UNORM** specifies a four-component, 32-bit unsigned normalized format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, an 8-bit R component in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_B8G8R8A8_SNORM** specifies a four-component, 32-bit signed normalized format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, an 8-bit R component in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_B8G8R8A8_USCALED** specifies a four-component, 32-bit unsigned scaled format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, an 8-bit R component in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_B8G8R8A8_SSCALED** specifies a four-component, 32-bit signed scaled format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, an 8-bit R component in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_B8G8R8A8_UINT** specifies a four-component, 32-bit unsigned integer format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, an 8-bit R component in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_B8G8R8A8_SINT** specifies a four-component, 32-bit signed integer format that has an 8-bit B component in byte 0, an 8-bit G component in byte 1, an 8-bit R component in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_B8G8R8A8_SRGB** specifies a four-component, 32-bit unsigned normalized format that has an 8-bit B component stored with sRGB nonlinear encoding in byte 0, an 8-bit G component stored with sRGB nonlinear encoding in byte 1, an 8-bit R component stored with sRGB nonlinear encoding in byte 2, and an 8-bit A component in byte 3.
- **VK_FORMAT_A8B8G8R8_UNORM_PACK32** specifies a four-component, 32-bit packed unsigned normalized format that has an 8-bit A component in bits 24..31, an 8-bit B component in bits 16..23, an 8-bit G component in bits 8..15, and an 8-bit R component in bits 0..7.

- **VK_FORMAT_A8B8G8R8_SNORM_PACK32** specifies a four-component, 32-bit packed signed normalized format that has an 8-bit A component in bits 24..31, an 8-bit B component in bits 16..23, an 8-bit G component in bits 8..15, and an 8-bit R component in bits 0..7.
- **VK_FORMAT_A8B8G8R8_USCALED_PACK32** specifies a four-component, 32-bit packed unsigned scaled integer format that has an 8-bit A component in bits 24..31, an 8-bit B component in bits 16..23, an 8-bit G component in bits 8..15, and an 8-bit R component in bits 0..7.
- **VK_FORMAT_A8B8G8R8_SSCALED_PACK32** specifies a four-component, 32-bit packed signed scaled integer format that has an 8-bit A component in bits 24..31, an 8-bit B component in bits 16..23, an 8-bit G component in bits 8..15, and an 8-bit R component in bits 0..7.
- **VK_FORMAT_A8B8G8R8_UINT_PACK32** specifies a four-component, 32-bit packed unsigned integer format that has an 8-bit A component in bits 24..31, an 8-bit B component in bits 16..23, an 8-bit G component in bits 8..15, and an 8-bit R component in bits 0..7.
- **VK_FORMAT_A8B8G8R8_SINT_PACK32** specifies a four-component, 32-bit packed signed integer format that has an 8-bit A component in bits 24..31, an 8-bit B component in bits 16..23, an 8-bit G component in bits 8..15, and an 8-bit R component in bits 0..7.
- **VK_FORMAT_A8B8G8R8_SRGB_PACK32** specifies a four-component, 32-bit packed unsigned normalized format that has an 8-bit A component in bits 24..31, an 8-bit B component stored with sRGB nonlinear encoding in bits 16..23, an 8-bit G component stored with sRGB nonlinear encoding in bits 8..15, and an 8-bit R component stored with sRGB nonlinear encoding in bits 0..7.
- **VK_FORMAT_A2R10G10B10_UNORM_PACK32** specifies a four-component, 32-bit packed unsigned normalized format that has a 2-bit A component in bits 30..31, a 10-bit R component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit B component in bits 0..9.
- **VK_FORMAT_A2R10G10B10_SNORM_PACK32** specifies a four-component, 32-bit packed signed normalized format that has a 2-bit A component in bits 30..31, a 10-bit R component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit B component in bits 0..9.
- **VK_FORMAT_A2R10G10B10_USCALED_PACK32** specifies a four-component, 32-bit packed unsigned scaled integer format that has a 2-bit A component in bits 30..31, a 10-bit R component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit B component in bits 0..9.
- **VK_FORMAT_A2R10G10B10_SSCALED_PACK32** specifies a four-component, 32-bit packed signed scaled integer format that has a 2-bit A component in bits 30..31, a 10-bit R component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit B component in bits 0..9.
- **VK_FORMAT_A2R10G10B10_UINT_PACK32** specifies a four-component, 32-bit packed unsigned integer format that has a 2-bit A component in bits 30..31, a 10-bit R component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit B component in bits 0..9.
- **VK_FORMAT_A2R10G10B10_SINT_PACK32** specifies a four-component, 32-bit packed signed integer format that has a 2-bit A component in bits 30..31, a 10-bit R component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit B component in bits 0..9.
- **VK_FORMAT_A2B10G10R10_UNORM_PACK32** specifies a four-component, 32-bit packed unsigned normalized format that has a 2-bit A component in bits 30..31, a 10-bit B component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit R component in bits 0..9.
- **VK_FORMAT_A2B10G10R10_SNORM_PACK32** specifies a four-component, 32-bit packed signed normalized format that has a 2-bit A component in bits 30..31, a 10-bit B component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit R component in bits 0..9.

- **VK_FORMAT_A2B10G10R10_USCALED_PACK32** specifies a four-component, 32-bit packed unsigned scaled integer format that has a 2-bit A component in bits 30..31, a 10-bit B component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit R component in bits 0..9.
- **VK_FORMAT_A2B10G10R10_SSCALED_PACK32** specifies a four-component, 32-bit packed signed scaled integer format that has a 2-bit A component in bits 30..31, a 10-bit B component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit R component in bits 0..9.
- **VK_FORMAT_A2B10G10R10_UINT_PACK32** specifies a four-component, 32-bit packed unsigned integer format that has a 2-bit A component in bits 30..31, a 10-bit B component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit R component in bits 0..9.
- **VK_FORMAT_A2B10G10R10_SINT_PACK32** specifies a four-component, 32-bit packed signed integer format that has a 2-bit A component in bits 30..31, a 10-bit B component in bits 20..29, a 10-bit G component in bits 10..19, and a 10-bit R component in bits 0..9.
- **VK_FORMAT_R16_UNORM** specifies a one-component, 16-bit unsigned normalized format that has a single 16-bit R component.
- **VK_FORMAT_R16_SNORM** specifies a one-component, 16-bit signed normalized format that has a single 16-bit R component.
- **VK_FORMAT_R16_USCALED** specifies a one-component, 16-bit unsigned scaled integer format that has a single 16-bit R component.
- **VK_FORMAT_R16_SSCALED** specifies a one-component, 16-bit signed scaled integer format that has a single 16-bit R component.
- **VK_FORMAT_R16_UINT** specifies a one-component, 16-bit unsigned integer format that has a single 16-bit R component.
- **VK_FORMAT_R16_SINT** specifies a one-component, 16-bit signed integer format that has a single 16-bit R component.
- **VK_FORMAT_R16_SFLOAT** specifies a one-component, 16-bit signed floating-point format that has a single 16-bit R component.
- **VK_FORMAT_R16G16_UNORM** specifies a two-component, 32-bit unsigned normalized format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- **VK_FORMAT_R16G16_SNORM** specifies a two-component, 32-bit signed normalized format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- **VK_FORMAT_R16G16_USCALED** specifies a two-component, 32-bit unsigned scaled integer format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- **VK_FORMAT_R16G16_SSCALED** specifies a two-component, 32-bit signed scaled integer format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- **VK_FORMAT_R16G16_UINT** specifies a two-component, 32-bit unsigned integer format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- **VK_FORMAT_R16G16_SINT** specifies a two-component, 32-bit signed integer format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- **VK_FORMAT_R16G16_SFLOAT** specifies a two-component, 32-bit signed floating-point format that has a 16-bit R component in bytes 0..1, and a 16-bit G component in bytes 2..3.
- **VK_FORMAT_R16G16B16_UNORM** specifies a three-component, 48-bit unsigned normalized format that

has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.

- **VK_FORMAT_R16G16B16_SNORM** specifies a three-component, 48-bit signed normalized format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- **VK_FORMAT_R16G16B16_USCALED** specifies a three-component, 48-bit unsigned scaled integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- **VK_FORMAT_R16G16B16_SSCALED** specifies a three-component, 48-bit signed scaled integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- **VK_FORMAT_R16G16B16_UINT** specifies a three-component, 48-bit unsigned integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- **VK_FORMAT_R16G16B16_SINT** specifies a three-component, 48-bit signed integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- **VK_FORMAT_R16G16B16_SFLOAT** specifies a three-component, 48-bit signed floating-point format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, and a 16-bit B component in bytes 4..5.
- **VK_FORMAT_R16G16B16A16_UNORM** specifies a four-component, 64-bit unsigned normalized format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- **VK_FORMAT_R16G16B16A16_SNORM** specifies a four-component, 64-bit signed normalized format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- **VK_FORMAT_R16G16B16A16_USCALED** specifies a four-component, 64-bit unsigned scaled integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- **VK_FORMAT_R16G16B16A16_SSCALED** specifies a four-component, 64-bit signed scaled integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- **VK_FORMAT_R16G16B16A16_UINT** specifies a four-component, 64-bit unsigned integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- **VK_FORMAT_R16G16B16A16_SINT** specifies a four-component, 64-bit signed integer format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- **VK_FORMAT_R16G16B16A16_SFLOAT** specifies a four-component, 64-bit signed floating-point format that has a 16-bit R component in bytes 0..1, a 16-bit G component in bytes 2..3, a 16-bit B component in bytes 4..5, and a 16-bit A component in bytes 6..7.
- **VK_FORMAT_R32_UINT** specifies a one-component, 32-bit unsigned integer format that has a single

32-bit R component.

- **VK_FORMAT_R32_SINT** specifies a one-component, 32-bit signed integer format that has a single 32-bit R component.
- **VK_FORMAT_R32_SFLOAT** specifies a one-component, 32-bit signed floating-point format that has a single 32-bit R component.
- **VK_FORMAT_R32G32_UINT** specifies a two-component, 64-bit unsigned integer format that has a 32-bit R component in bytes 0..3, and a 32-bit G component in bytes 4..7.
- **VK_FORMAT_R32G32_SINT** specifies a two-component, 64-bit signed integer format that has a 32-bit R component in bytes 0..3, and a 32-bit G component in bytes 4..7.
- **VK_FORMAT_R32G32_SFLOAT** specifies a two-component, 64-bit signed floating-point format that has a 32-bit R component in bytes 0..3, and a 32-bit G component in bytes 4..7.
- **VK_FORMAT_R32G32B32_UINT** specifies a three-component, 96-bit unsigned integer format that has a 32-bit R component in bytes 0..3, a 32-bit G component in bytes 4..7, and a 32-bit B component in bytes 8..11.
- **VK_FORMAT_R32G32B32_SINT** specifies a three-component, 96-bit signed integer format that has a 32-bit R component in bytes 0..3, a 32-bit G component in bytes 4..7, and a 32-bit B component in bytes 8..11.
- **VK_FORMAT_R32G32B32_SFLOAT** specifies a three-component, 96-bit signed floating-point format that has a 32-bit R component in bytes 0..3, a 32-bit G component in bytes 4..7, and a 32-bit B component in bytes 8..11.
- **VK_FORMAT_R32G32B32A32_UINT** specifies a four-component, 128-bit unsigned integer format that has a 32-bit R component in bytes 0..3, a 32-bit G component in bytes 4..7, a 32-bit B component in bytes 8..11, and a 32-bit A component in bytes 12..15.
- **VK_FORMAT_R32G32B32A32_SINT** specifies a four-component, 128-bit signed integer format that has a 32-bit R component in bytes 0..3, a 32-bit G component in bytes 4..7, a 32-bit B component in bytes 8..11, and a 32-bit A component in bytes 12..15.
- **VK_FORMAT_R32G32B32A32_SFLOAT** specifies a four-component, 128-bit signed floating-point format that has a 32-bit R component in bytes 0..3, a 32-bit G component in bytes 4..7, a 32-bit B component in bytes 8..11, and a 32-bit A component in bytes 12..15.
- **VK_FORMAT_R64_UINT** specifies a one-component, 64-bit unsigned integer format that has a single 64-bit R component.
- **VK_FORMAT_R64_SINT** specifies a one-component, 64-bit signed integer format that has a single 64-bit R component.
- **VK_FORMAT_R64_SFLOAT** specifies a one-component, 64-bit signed floating-point format that has a single 64-bit R component.
- **VK_FORMAT_R64G64_UINT** specifies a two-component, 128-bit unsigned integer format that has a 64-bit R component in bytes 0..7, and a 64-bit G component in bytes 8..15.
- **VK_FORMAT_R64G64_SINT** specifies a two-component, 128-bit signed integer format that has a 64-bit R component in bytes 0..7, and a 64-bit G component in bytes 8..15.
- **VK_FORMAT_R64G64_SFLOAT** specifies a two-component, 128-bit signed floating-point format that has a 64-bit R component in bytes 0..7, and a 64-bit G component in bytes 8..15.

- **VK_FORMAT_R64G64B64_UINT** specifies a three-component, 192-bit unsigned integer format that has a 64-bit R component in bytes 0..7, a 64-bit G component in bytes 8..15, and a 64-bit B component in bytes 16..23.
- **VK_FORMAT_R64G64B64_SINT** specifies a three-component, 192-bit signed integer format that has a 64-bit R component in bytes 0..7, a 64-bit G component in bytes 8..15, and a 64-bit B component in bytes 16..23.
- **VK_FORMAT_R64G64B64_SFLOAT** specifies a three-component, 192-bit signed floating-point format that has a 64-bit R component in bytes 0..7, a 64-bit G component in bytes 8..15, and a 64-bit B component in bytes 16..23.
- **VK_FORMAT_R64G64B64A64_UINT** specifies a four-component, 256-bit unsigned integer format that has a 64-bit R component in bytes 0..7, a 64-bit G component in bytes 8..15, a 64-bit B component in bytes 16..23, and a 64-bit A component in bytes 24..31.
- **VK_FORMAT_R64G64B64A64_SINT** specifies a four-component, 256-bit signed integer format that has a 64-bit R component in bytes 0..7, a 64-bit G component in bytes 8..15, a 64-bit B component in bytes 16..23, and a 64-bit A component in bytes 24..31.
- **VK_FORMAT_R64G64B64A64_SFLOAT** specifies a four-component, 256-bit signed floating-point format that has a 64-bit R component in bytes 0..7, a 64-bit G component in bytes 8..15, a 64-bit B component in bytes 16..23, and a 64-bit A component in bytes 24..31.
- **VK_FORMAT_B10G11R11_UFLOAT_PACK32** specifies a three-component, 32-bit packed unsigned floating-point format that has a 10-bit B component in bits 22..31, an 11-bit G component in bits 11..21, an 11-bit R component in bits 0..10. See [Unsigned 10-Bit Floating-Point Numbers](#) and [Unsigned 11-Bit Floating-Point Numbers](#).
- **VK_FORMAT_E5B9G9R9_UFLOAT_PACK32** specifies a three-component, 32-bit packed unsigned floating-point format that has a 5-bit shared exponent in bits 27..31, a 9-bit B component mantissa in bits 18..26, a 9-bit G component mantissa in bits 9..17, and a 9-bit R component mantissa in bits 0..8.
- **VK_FORMAT_D16_UNORM** specifies a one-component, 16-bit unsigned normalized format that has a single 16-bit depth component.
- **VK_FORMAT_X8_D24_UNORM_PACK32** specifies a two-component, 32-bit format that has 24 unsigned normalized bits in the depth component and, **optionally**, 8 bits that are unused.
- **VK_FORMAT_D32_SFLOAT** specifies a one-component, 32-bit signed floating-point format that has 32 bits in the depth component.
- **VK_FORMAT_S8_UINT** specifies a one-component, 8-bit unsigned integer format that has 8 bits in the stencil component.
- **VK_FORMAT_D16_UNORM_S8_UINT** specifies a two-component, 24-bit format that has 16 unsigned normalized bits in the depth component and 8 unsigned integer bits in the stencil component.
- **VK_FORMAT_D24_UNORM_S8_UINT** specifies a two-component, 32-bit packed format that has 8 unsigned integer bits in the stencil component, and 24 unsigned normalized bits in the depth component.
- **VK_FORMAT_D32_SFLOAT_S8_UINT** specifies a two-component format that has 32 signed float bits in the depth component and 8 unsigned integer bits in the stencil component. There are **optionally** 24 bits that are unused.
- **VK_FORMAT_BC1_RGB_UNORM_BLOCK** specifies a three-component, block-compressed format where

each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data. This format has no alpha and is considered opaque.

- **VK_FORMAT_BC1_RGB_SRGB_BLOCK** specifies a three-component, block-compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data with sRGB nonlinear encoding. This format has no alpha and is considered opaque.
- **VK_FORMAT_BC1_RGBA_UNORM_BLOCK** specifies a four-component, block-compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data, and provides 1 bit of alpha.
- **VK_FORMAT_BC1_RGBA_SRGB_BLOCK** specifies a four-component, block-compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data with sRGB nonlinear encoding, and provides 1 bit of alpha.
- **VK_FORMAT_BC2_UNORM_BLOCK** specifies a four-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with the first 64 bits encoding alpha values followed by 64 bits encoding RGB values.
- **VK_FORMAT_BC2_SRGB_BLOCK** specifies a four-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with the first 64 bits encoding alpha values followed by 64 bits encoding RGB values with sRGB nonlinear encoding.
- **VK_FORMAT_BC3_UNORM_BLOCK** specifies a four-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with the first 64 bits encoding alpha values followed by 64 bits encoding RGB values.
- **VK_FORMAT_BC3_SRGB_BLOCK** specifies a four-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with the first 64 bits encoding alpha values followed by 64 bits encoding RGB values with sRGB nonlinear encoding.
- **VK_FORMAT_BC4_UNORM_BLOCK** specifies a one-component, block-compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized red texel data.
- **VK_FORMAT_BC4_SNORM_BLOCK** specifies a one-component, block-compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of signed normalized red texel data.
- **VK_FORMAT_BC5_UNORM_BLOCK** specifies a two-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RG texel data with the first 64 bits encoding red values followed by 64 bits encoding green values.
- **VK_FORMAT_BC5_SNORM_BLOCK** specifies a two-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of signed normalized RG texel data with the first 64 bits encoding red values followed by 64 bits encoding green values.
- **VK_FORMAT_BC6H_UFLOAT_BLOCK** specifies a three-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned floating-point RGB texel data.
- **VK_FORMAT_BC6H_SFLOAT_BLOCK** specifies a three-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of signed floating-point RGB texel data.
- **VK_FORMAT_BC7_UNORM_BLOCK** specifies a four-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel

data.

- **VK_FORMAT_BC7_SRGB_BLOCK** specifies a four-component, block-compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK** specifies a three-component, ETC2 compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data. This format has no alpha and is considered opaque.
- **VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK** specifies a three-component, ETC2 compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data with sRGB nonlinear encoding. This format has no alpha and is considered opaque.
- **VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK** specifies a four-component, ETC2 compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data, and provides 1 bit of alpha.
- **VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK** specifies a four-component, ETC2 compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGB texel data with sRGB nonlinear encoding, and provides 1 bit of alpha.
- **VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK** specifies a four-component, ETC2 compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with the first 64 bits encoding alpha values followed by 64 bits encoding RGB values.
- **VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK** specifies a four-component, ETC2 compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with the first 64 bits encoding alpha values followed by 64 bits encoding RGB values with sRGB nonlinear encoding applied.
- **VK_FORMAT_EAC_R11_UNORM_BLOCK** specifies a one-component, ETC2 compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized red texel data.
- **VK_FORMAT_EAC_R11_SNORM_BLOCK** specifies a one-component, ETC2 compressed format where each 64-bit compressed texel block encodes a 4×4 rectangle of signed normalized red texel data.
- **VK_FORMAT_EAC_R11G11_UNORM_BLOCK** specifies a two-component, ETC2 compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RG texel data with the first 64 bits encoding red values followed by 64 bits encoding green values.
- **VK_FORMAT_EAC_R11G11_SNORM_BLOCK** specifies a two-component, ETC2 compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of signed normalized RG texel data with the first 64 bits encoding red values followed by 64 bits encoding green values.
- **VK_FORMAT_ASTC_4x4_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_4x4_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 4×4 rectangle of signed floating-point RGBA texel

data.

- **VK_FORMAT_ASTC_5x4_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 5×4 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_5x4_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 5×4 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 5×4 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_ASTC_5x5_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 5×5 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_5x5_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 5×5 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 5×5 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_ASTC_6x5_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 6×5 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_6x5_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 6×5 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 6×5 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_ASTC_6x6_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 6×6 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_6x6_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 6×6 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 6×6 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_ASTC_8x5_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes an 8×5 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_8x5_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes an 8×5 rectangle of unsigned normalized RGBA

texel data with sRGB nonlinear encoding applied to the RGB components.

- **VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 8×5 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_ASTC_8x6_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes an 8×6 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_8x6_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes an 8×6 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 8×6 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_ASTC_8x8_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes an 8×8 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_8x8_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes an 8×8 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 8×8 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_ASTC_10x5_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×5 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_10x5_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×5 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×5 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_ASTC_10x6_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×6 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_10x6_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×6 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×6 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_ASTC_10x8_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×8 rectangle of unsigned normalized RGBA

texel data.

- **VK_FORMAT_ASTC_10x8_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×8 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×8 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_ASTC_10x10_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×10 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_10x10_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×10 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 10×10 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_ASTC_12x10_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 12×10 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_12x10_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 12×10 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 12×10 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_ASTC_12x12_UNORM_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 12×12 rectangle of unsigned normalized RGBA texel data.
- **VK_FORMAT_ASTC_12x12_SRGB_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 12×12 rectangle of unsigned normalized RGBA texel data with sRGB nonlinear encoding applied to the RGB components.
- **VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK** specifies a four-component, ASTC compressed format where each 128-bit compressed texel block encodes a 12×12 rectangle of signed floating-point RGBA texel data.
- **VK_FORMAT_G8B8G8R8_422_UNORM** specifies a four-component, 32-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each *i* coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has an 8-bit G component for the even *i* coordinate in byte 0, an 8-bit B component in byte 1, an 8-bit G component for the odd *i* coordinate in byte 2, and an 8-bit R component in byte 3. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.

- **VK_FORMAT_B8G8R8G8_422_UNORM** specifies a four-component, 32-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each i coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has an 8-bit B component in byte 0, an 8-bit G component for the even i coordinate in byte 1, an 8-bit R component in byte 2, and an 8-bit G component for the odd i coordinate in byte 3. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.
- **VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM** specifies an unsigned normalized *multi-planar format* that has an 8-bit G component in plane 0, an 8-bit B component in plane 1, and an 8-bit R component in plane 2. The horizontal and vertical dimensions of the R and B planes are halved relative to the image dimensions, and each R and B component is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using **VK_IMAGE_ASPECT_PLANE_0_BIT** for the G plane, **VK_IMAGE_ASPECT_PLANE_1_BIT** for the B plane, and **VK_IMAGE_ASPECT_PLANE_2_BIT** for the R plane. This format only supports images with a width and height that is a multiple of two.
- **VK_FORMAT_G8_B8R8_2PLANE_420_UNORM** specifies an unsigned normalized *multi-planar format* that has an 8-bit G component in plane 0, and a two-component, 16-bit BR plane 1 consisting of an 8-bit B component in byte 0 and an 8-bit R component in byte 1. The horizontal and vertical dimensions of the BR plane are halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using **VK_IMAGE_ASPECT_PLANE_0_BIT** for the G plane, and **VK_IMAGE_ASPECT_PLANE_1_BIT** for the BR plane. This format only supports images with a width and height that is a multiple of two.
- **VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM** specifies an unsigned normalized *multi-planar format* that has an 8-bit G component in plane 0, an 8-bit B component in plane 1, and an 8-bit R component in plane 2. The horizontal dimension of the R and B plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using **VK_IMAGE_ASPECT_PLANE_0_BIT** for the G plane, **VK_IMAGE_ASPECT_PLANE_1_BIT** for the B plane, and **VK_IMAGE_ASPECT_PLANE_2_BIT** for the R plane. This format only supports images with a width that is a multiple of two.
- **VK_FORMAT_G8_B8R8_2PLANE_422_UNORM** specifies an unsigned normalized *multi-planar format* that has an 8-bit G component in plane 0, and a two-component, 16-bit BR plane 1 consisting of an 8-bit B component in byte 0 and an 8-bit R component in byte 1. The horizontal dimension of the BR plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using **VK_IMAGE_ASPECT_PLANE_0_BIT** for the G plane, and **VK_IMAGE_ASPECT_PLANE_1_BIT** for the BR plane. This format only supports images with a width that is a multiple of two.
- **VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM** specifies an unsigned normalized *multi-planar format* that has an 8-bit G component in plane 0, an 8-bit B component in plane 1, and an 8-bit R component in plane 2. Each plane has the same dimensions and each R, G and B component contributes to a

single texel. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, `VK_IMAGE_ASPECT_PLANE_1_BIT` for the B plane, and `VK_IMAGE_ASPECT_PLANE_2_BIT` for the R plane.

- `VK_FORMAT_R10X6_UNORM_PACK16` specifies a one-component, 16-bit unsigned normalized format that has a single 10-bit R component in the top 10 bits of a 16-bit word, with the bottom 6 bits unused.
- `VK_FORMAT_R10X6G10X6_UNORM_2PACK16` specifies a two-component, 32-bit unsigned normalized format that has a 10-bit R component in the top 10 bits of the word in bytes 0..1, and a 10-bit G component in the top 10 bits of the word in bytes 2..3, with the bottom 6 bits of each word unused.
- `VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16` specifies a four-component, 64-bit unsigned normalized format that has a 10-bit R component in the top 10 bits of the word in bytes 0..1, a 10-bit G component in the top 10 bits of the word in bytes 2..3, a 10-bit B component in the top 10 bits of the word in bytes 4..5, and a 10-bit A component in the top 10 bits of the word in bytes 6..7, with the bottom 6 bits of each word unused.
- `VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16` specifies a four-component, 64-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each i coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has a 10-bit G component for the even i coordinate in the top 10 bits of the word in bytes 0..1, a 10-bit B component in the top 10 bits of the word in bytes 2..3, a 10-bit G component for the odd i coordinate in the top 10 bits of the word in bytes 4..5, and a 10-bit R component in the top 10 bits of the word in bytes 6..7, with the bottom 6 bits of each word unused. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.
- `VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16` specifies a four-component, 64-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each i coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has a 10-bit B component in the top 10 bits of the word in bytes 0..1, a 10-bit G component for the even i coordinate in the top 10 bits of the word in bytes 2..3, a 10-bit R component in the top 10 bits of the word in bytes 4..5, and a 10-bit G component for the odd i coordinate in the top 10 bits of the word in bytes 6..7, with the bottom 6 bits of each word unused. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.
- `VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 10-bit G component in the top 10 bits of each 16-bit word of plane 0, a 10-bit B component in the top 10 bits of each 16-bit word of plane 1, and a 10-bit R component in the top 10 bits of each 16-bit word of plane 2, with the bottom 6 bits of each word unused. The horizontal and vertical dimensions of the R and B planes are halved relative to the image dimensions, and each R and B component is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using

`VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, `VK_IMAGE_ASPECT_PLANE_1_BIT` for the B plane, and `VK_IMAGE_ASPECT_PLANE_2_BIT` for the R plane. This format only supports images with a width and height that is a multiple of two.

- `VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 10-bit G component in the top 10 bits of each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 10-bit B component in the top 10 bits of the word in bytes 0..1, and a 10-bit R component in the top 10 bits of the word in bytes 2..3, with the bottom 6 bits of each word unused. The horizontal and vertical dimensions of the BR plane are halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, and `VK_IMAGE_ASPECT_PLANE_1_BIT` for the BR plane. This format only supports images with a width and height that is a multiple of two.
- `VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 10-bit G component in the top 10 bits of each 16-bit word of plane 0, a 10-bit B component in the top 10 bits of each 16-bit word of plane 1, and a 10-bit R component in the top 10 bits of each 16-bit word of plane 2, with the bottom 6 bits of each word unused. The horizontal dimension of the R and B plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, `VK_IMAGE_ASPECT_PLANE_1_BIT` for the B plane, and `VK_IMAGE_ASPECT_PLANE_2_BIT` for the R plane. This format only supports images with a width that is a multiple of two.
- `VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 10-bit G component in the top 10 bits of each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 10-bit B component in the top 10 bits of the word in bytes 0..1, and a 10-bit R component in the top 10 bits of the word in bytes 2..3, with the bottom 6 bits of each word unused. The horizontal dimension of the BR plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, and `VK_IMAGE_ASPECT_PLANE_1_BIT` for the BR plane. This format only supports images with a width that is a multiple of two.
- `VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 10-bit G component in the top 10 bits of each 16-bit word of plane 0, a 10-bit B component in the top 10 bits of each 16-bit word of plane 1, and a 10-bit R component in the top 10 bits of each 16-bit word of plane 2, with the bottom 6 bits of each word unused. Each plane has the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, `VK_IMAGE_ASPECT_PLANE_1_BIT` for the B plane, and `VK_IMAGE_ASPECT_PLANE_2_BIT` for the R plane.
- `VK_FORMAT_R12X4_UNORM_PACK16` specifies a one-component, 16-bit unsigned normalized format that has a single 12-bit R component in the top 12 bits of a 16-bit word, with the bottom 4 bits unused.
- `VK_FORMAT_R12X4G12X4_UNORM_2PACK16` specifies a two-component, 32-bit unsigned normalized

format that has a 12-bit R component in the top 12 bits of the word in bytes 0..1, and a 12-bit G component in the top 12 bits of the word in bytes 2..3, with the bottom 4 bits of each word unused.

- **VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16** specifies a four-component, 64-bit unsigned normalized format that has a 12-bit R component in the top 12 bits of the word in bytes 0..1, a 12-bit G component in the top 12 bits of the word in bytes 2..3, a 12-bit B component in the top 12 bits of the word in bytes 4..5, and a 12-bit A component in the top 12 bits of the word in bytes 6..7, with the bottom 4 bits of each word unused.
- **VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16** specifies a four-component, 64-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each i coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has a 12-bit G component for the even i coordinate in the top 12 bits of the word in bytes 0..1, a 12-bit B component in the top 12 bits of the word in bytes 2..3, a 12-bit G component for the odd i coordinate in the top 12 bits of the word in bytes 4..5, and a 12-bit R component in the top 12 bits of the word in bytes 6..7, with the bottom 4 bits of each word unused. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.
- **VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16** specifies a four-component, 64-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each i coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has a 12-bit B component in the top 12 bits of the word in bytes 0..1, a 12-bit G component for the even i coordinate in the top 12 bits of the word in bytes 2..3, a 12-bit R component in the top 12 bits of the word in bytes 4..5, and a 12-bit G component for the odd i coordinate in the top 12 bits of the word in bytes 6..7, with the bottom 4 bits of each word unused. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.
- **VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16** specifies an unsigned normalized *multi-planar format* that has a 12-bit G component in the top 12 bits of each 16-bit word of plane 0, a 12-bit B component in the top 12 bits of each 16-bit word of plane 1, and a 12-bit R component in the top 12 bits of each 16-bit word of plane 2, with the bottom 4 bits of each word unused. The horizontal and vertical dimensions of the R and B planes are halved relative to the image dimensions, and each R and B component is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using **VK_IMAGE_ASPECT_PLANE_0_BIT** for the G plane, **VK_IMAGE_ASPECT_PLANE_1_BIT** for the B plane, and **VK_IMAGE_ASPECT_PLANE_2_BIT** for the R plane. This format only supports images with a width and height that is a multiple of two.
- **VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16** specifies an unsigned normalized *multi-planar format* that has a 12-bit G component in the top 12 bits of each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 12-bit B component in the top 12 bits of the word in bytes 0..1, and a 12-bit R component in the top 12 bits of the word in bytes 2..3, with the bottom 4 bits of each word unused. The horizontal and vertical dimensions of the BR plane

are halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, and `VK_IMAGE_ASPECT_PLANE_1_BIT` for the BR plane. This format only supports images with a width and height that is a multiple of two.

- `VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 12-bit G component in the top 12 bits of each 16-bit word of plane 0, a 12-bit B component in the top 12 bits of each 16-bit word of plane 1, and a 12-bit R component in the top 12 bits of each 16-bit word of plane 2, with the bottom 4 bits of each word unused. The horizontal dimension of the R and B plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, `VK_IMAGE_ASPECT_PLANE_1_BIT` for the B plane, and `VK_IMAGE_ASPECT_PLANE_2_BIT` for the R plane. This format only supports images with a width that is a multiple of two.
- `VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 12-bit G component in the top 12 bits of each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 12-bit B component in the top 12 bits of the word in bytes 0..1, and a 12-bit R component in the top 12 bits of the word in bytes 2..3, with the bottom 4 bits of each word unused. The horizontal dimension of the BR plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, and `VK_IMAGE_ASPECT_PLANE_1_BIT` for the BR plane. This format only supports images with a width that is a multiple of two.
- `VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 12-bit G component in the top 12 bits of each 16-bit word of plane 0, a 12-bit B component in the top 12 bits of each 16-bit word of plane 1, and a 12-bit R component in the top 12 bits of each 16-bit word of plane 2, with the bottom 4 bits of each word unused. Each plane has the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, `VK_IMAGE_ASPECT_PLANE_1_BIT` for the B plane, and `VK_IMAGE_ASPECT_PLANE_2_BIT` for the R plane.
- `VK_FORMAT_G16B16G16R16_422_UNORM` specifies a four-component, 64-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each i coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the image. This format has a 16-bit G component for the even i coordinate in the word in bytes 0..1, a 16-bit B component in the word in bytes 2..3, a 16-bit G component for the odd i coordinate in the word in bytes 4..5, and a 16-bit R component in the word in bytes 6..7. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.
- `VK_FORMAT_B16G16R16G16_422_UNORM` specifies a four-component, 64-bit format containing a pair of G components, an R component, and a B component, collectively encoding a 2×1 rectangle of unsigned normalized RGB texel data. One G value is present at each i coordinate, with the B and R values shared across both G values and thus recorded at half the horizontal resolution of the

image. This format has a 16-bit B component in the word in bytes 0..1, a 16-bit G component for the even i coordinate in the word in bytes 2..3, a 16-bit R component in the word in bytes 4..5, and a 16-bit G component for the odd i coordinate in the word in bytes 6..7. This format only supports images with a width that is a multiple of two. For the purposes of the constraints on copy extents, this format is treated as a compressed format with a 2×1 compressed texel block.

- **VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM** specifies an unsigned normalized *multi-planar format* that has a 16-bit G component in each 16-bit word of plane 0, a 16-bit B component in each 16-bit word of plane 1, and a 16-bit R component in each 16-bit word of plane 2. The horizontal and vertical dimensions of the R and B planes are halved relative to the image dimensions, and each R and B component is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using **VK_IMAGE_ASPECT_PLANE_0_BIT** for the G plane, **VK_IMAGE_ASPECT_PLANE_1_BIT** for the B plane, and **VK_IMAGE_ASPECT_PLANE_2_BIT** for the R plane. This format only supports images with a width and height that is a multiple of two.
- **VK_FORMAT_G16_B16R16_2PLANE_420_UNORM** specifies an unsigned normalized *multi-planar format* that has a 16-bit G component in each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 16-bit B component in the word in bytes 0..1, and a 16-bit R component in the word in bytes 2..3. The horizontal and vertical dimensions of the BR plane are halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$ and $\lfloor j_G \times 0.5 \rfloor = j_B = j_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using **VK_IMAGE_ASPECT_PLANE_0_BIT** for the G plane, and **VK_IMAGE_ASPECT_PLANE_1_BIT** for the BR plane. This format only supports images with a width and height that is a multiple of two.
- **VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM** specifies an unsigned normalized *multi-planar format* that has a 16-bit G component in each 16-bit word of plane 0, a 16-bit B component in each 16-bit word of plane 1, and a 16-bit R component in each 16-bit word of plane 2. The horizontal dimension of the R and B plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using **VK_IMAGE_ASPECT_PLANE_0_BIT** for the G plane, **VK_IMAGE_ASPECT_PLANE_1_BIT** for the B plane, and **VK_IMAGE_ASPECT_PLANE_2_BIT** for the R plane. This format only supports images with a width that is a multiple of two.
- **VK_FORMAT_G16_B16R16_2PLANE_422_UNORM** specifies an unsigned normalized *multi-planar format* that has a 16-bit G component in each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 16-bit B component in the word in bytes 0..1, and a 16-bit R component in the word in bytes 2..3. The horizontal dimension of the BR plane is halved relative to the image dimensions, and each R and B value is shared with the G components for which $\lfloor i_G \times 0.5 \rfloor = i_B = i_R$. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using **VK_IMAGE_ASPECT_PLANE_0_BIT** for the G plane, and **VK_IMAGE_ASPECT_PLANE_1_BIT** for the BR plane. This format only supports images with a width that is a multiple of two.
- **VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM** specifies an unsigned normalized *multi-planar format* that has a 16-bit G component in each 16-bit word of plane 0, a 16-bit B component in each 16-bit word of plane 1, and a 16-bit R component in each 16-bit word of plane 2. Each plane has the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via

[vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, `VK_IMAGE_ASPECT_PLANE_1_BIT` for the B plane, and `VK_IMAGE_ASPECT_PLANE_2_BIT` for the R plane.

- `VK_FORMAT_G8_B8R8_2PLANE_444_UNORM` specifies an unsigned normalized *multi-planar format* that has an 8-bit G component in plane 0, and a two-component, 16-bit BR plane 1 consisting of an 8-bit B component in byte 0 and an 8-bit R component in byte 1. Both planes have the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, and `VK_IMAGE_ASPECT_PLANE_1_BIT` for the BR plane.
- `VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 10-bit G component in the top 10 bits of each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 10-bit B component in the top 10 bits of the word in bytes 0..1, and a 10-bit R component in the top 10 bits of the word in bytes 2..3, the bottom 6 bits of each word unused. Both planes have the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, and `VK_IMAGE_ASPECT_PLANE_1_BIT` for the BR plane.
- `VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16` specifies an unsigned normalized *multi-planar format* that has a 12-bit G component in the top 12 bits of each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 12-bit B component in the top 12 bits of the word in bytes 0..1, and a 12-bit R component in the top 12 bits of the word in bytes 2..3, the bottom 4 bits of each word unused. Both planes have the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, and `VK_IMAGE_ASPECT_PLANE_1_BIT` for the BR plane.
- `VK_FORMAT_G16_B16R16_2PLANE_444_UNORM` specifies an unsigned normalized *multi-planar format* that has a 16-bit G component in each 16-bit word of plane 0, and a two-component, 32-bit BR plane 1 consisting of a 16-bit B component in the word in bytes 0..1, and a 16-bit R component in the word in bytes 2..3. Both planes have the same dimensions and each R, G and B component contributes to a single texel. The location of each plane when this image is in linear layout can be determined via [vkGetImageSubresourceLayout](#), using `VK_IMAGE_ASPECT_PLANE_0_BIT` for the G plane, and `VK_IMAGE_ASPECT_PLANE_1_BIT` for the BR plane.

35.1.1. Compatible Formats of Planes of Multi-Planar Formats

Individual planes of [multi-planar formats](#) are size-compatible with single-plane color formats if they occupy the same number of bits per texel block, and are compatible with those formats if they have the same block extent.

In the following table, individual planes of a *multi-planar* format are compatible with the format listed against the relevant plane index for that multi-planar format, and any format compatible with the listed single-plane format according to [Format Compatibility Classes](#). These planes are also [size-compatible](#) with any format that is [size-compatible](#) with the listed single-plane format.

Table 50. Plane Format Compatibility Table

Plane	Compatible format for plane	Width relative to the width w of the plane with the largest dimensions	Height relative to the height h of the plane with the largest dimensions
VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM			
0	VK_FORMAT_R8_UNORM	w	h
1	VK_FORMAT_R8_UNORM	$w/2$	$h/2$
2	VK_FORMAT_R8_UNORM	$w/2$	$h/2$
VK_FORMAT_G8_B8R8_2PLANE_420_UNORM			
0	VK_FORMAT_R8_UNORM	w	h
1	VK_FORMAT_R8G8_UNORM	$w/2$	$h/2$
VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM			
0	VK_FORMAT_R8_UNORM	w	h
1	VK_FORMAT_R8_UNORM	$w/2$	h
2	VK_FORMAT_R8_UNORM	$w/2$	h
VK_FORMAT_G8_B8R8_2PLANE_422_UNORM			
0	VK_FORMAT_R8_UNORM	w	h
1	VK_FORMAT_R8G8_UNORM	$w/2$	h
VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM			
0	VK_FORMAT_R8_UNORM	w	h
1	VK_FORMAT_R8_UNORM	w	h
2	VK_FORMAT_R8_UNORM	w	h
VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16			
0	VK_FORMAT_R10X6_UNORM_PACK16	w	h
1	VK_FORMAT_R10X6_UNORM_PACK16	$w/2$	$h/2$
2	VK_FORMAT_R10X6_UNORM_PACK16	$w/2$	$h/2$
VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16			
0	VK_FORMAT_R10X6_UNORM_PACK16	w	h
1	VK_FORMAT_R10X6G10X6_UNORM_2PACK16	$w/2$	$h/2$
VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16			
0	VK_FORMAT_R10X6_UNORM_PACK16	w	h
1	VK_FORMAT_R10X6_UNORM_PACK16	$w/2$	h
2	VK_FORMAT_R10X6_UNORM_PACK16	$w/2$	h
VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16			
0	VK_FORMAT_R10X6_UNORM_PACK16	w	h
1	VK_FORMAT_R10X6G10X6_UNORM_2PACK16	$w/2$	h

Plane	Compatible format for plane	Width relative to the width w of the plane with the largest dimensions	Height relative to the height h of the plane with the largest dimensions
VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16			
0	VK_FORMAT_R10X6_UNORM_PACK16	w	h
1	VK_FORMAT_R10X6_UNORM_PACK16	w	h
2	VK_FORMAT_R10X6_UNORM_PACK16	w	h
VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16			
0	VK_FORMAT_R12X4_UNORM_PACK16	w	h
1	VK_FORMAT_R12X4_UNORM_PACK16	$w/2$	$h/2$
2	VK_FORMAT_R12X4_UNORM_PACK16	$w/2$	$h/2$
VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16			
0	VK_FORMAT_R12X4_UNORM_PACK16	w	h
1	VK_FORMAT_R12X4G12X4_UNORM_2PACK16	$w/2$	$h/2$
VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16			
0	VK_FORMAT_R12X4_UNORM_PACK16	w	h
1	VK_FORMAT_R12X4_UNORM_PACK16	$w/2$	h
2	VK_FORMAT_R12X4_UNORM_PACK16	$w/2$	h
VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16			
0	VK_FORMAT_R12X4_UNORM_PACK16	w	h
1	VK_FORMAT_R12X4G12X4_UNORM_2PACK16	$w/2$	h
VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16			
0	VK_FORMAT_R12X4_UNORM_PACK16	w	h
1	VK_FORMAT_R12X4_UNORM_PACK16	w	h
2	VK_FORMAT_R12X4_UNORM_PACK16	w	h
VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM			
0	VK_FORMAT_R16_UNORM	w	h
1	VK_FORMAT_R16_UNORM	$w/2$	$h/2$
2	VK_FORMAT_R16_UNORM	$w/2$	$h/2$
VK_FORMAT_G16_B16R16_2PLANE_420_UNORM			
0	VK_FORMAT_R16_UNORM	w	h
1	VK_FORMAT_R16G16_UNORM	$w/2$	$h/2$
VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM			
0	VK_FORMAT_R16_UNORM	w	h
1	VK_FORMAT_R16_UNORM	$w/2$	h

Plane	Compatible format for plane	Width relative to the width w of the plane with the largest dimensions	Height relative to the height h of the plane with the largest dimensions
2	VK_FORMAT_R16_UNORM	$w/2$	h
VK_FORMAT_G16_B16R16_2PLANE_422_UNORM			
0	VK_FORMAT_R16_UNORM	w	h
1	VK_FORMAT_R16G16_UNORM	$w/2$	h
VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM			
0	VK_FORMAT_R16_UNORM	w	h
1	VK_FORMAT_R16_UNORM	w	h
2	VK_FORMAT_R16_UNORM	w	h
VK_FORMAT_G8_B8R8_2PLANE_444_UNORM			
0	VK_FORMAT_R8_UNORM	w	h
1	VK_FORMAT_R8G8_UNORM	w	h
VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16			
0	VK_FORMAT_R10X6_UNORM_PACK16	w	h
1	VK_FORMAT_R10X6G10X6_UNORM_2PACK16	w	h
VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16			
0	VK_FORMAT_R12X4_UNORM_PACK16	w	h
1	VK_FORMAT_R12X4G12X4_UNORM_2PACK16	w	h
VK_FORMAT_G16_B16R16_2PLANE_444_UNORM			
0	VK_FORMAT_R16_UNORM	w	h
1	VK_FORMAT_R16G16_UNORM	w	h

35.1.2. Multi-Planar Format Image Aspect

When using [VkImageAspectFlagBits](#) to select a plane of a [multi-planar format](#), the following are the valid options:

- Two planes
 - VK_IMAGE_ASPECT_PLANE_0_BIT
 - VK_IMAGE_ASPECT_PLANE_1_BIT
- Three planes
 - VK_IMAGE_ASPECT_PLANE_0_BIT
 - VK_IMAGE_ASPECT_PLANE_1_BIT
 - VK_IMAGE_ASPECT_PLANE_2_BIT

35.1.3. Packed Formats

For the purposes of address alignment when accessing buffer memory containing vertex attribute or texel data, the following formats are considered *packed* - components of the texels or attributes are stored in bitfields packed into one or more 8-, 16-, or 32-bit fundamental data type.

- **Packed into 8-bit data types:**
 - `VK_FORMAT_R4G4_UNORM_PACK8`
- **Packed into 16-bit data types:**
 - `VK_FORMAT_R4G4B4A4_UNORM_PACK16`
 - `VK_FORMAT_B4G4R4A4_UNORM_PACK16`
 - `VK_FORMAT_R5G6B5_UNORM_PACK16`
 - `VK_FORMAT_B5G6R5_UNORM_PACK16`
 - `VK_FORMAT_R5G5B5A1_UNORM_PACK16`
 - `VK_FORMAT_B5G5R5A1_UNORM_PACK16`
 - `VK_FORMAT_A1R5G5B5_UNORM_PACK16`
 - `VK_FORMAT_R10X6_UNORM_PACK16`
 - `VK_FORMAT_R10X6G10X6_UNORM_2PACK16`
 - `VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16`
 - `VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16`
 - `VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16`
 - `VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16`
 - `VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16`
 - `VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16`
 - `VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16`
 - `VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16`
 - `VK_FORMAT_R12X4_UNORM_PACK16`
 - `VK_FORMAT_R12X4G12X4_UNORM_2PACK16`
 - `VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16`
 - `VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16`
 - `VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16`
 - `VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16`
 - `VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16`
 - `VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16`
 - `VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16`
 - `VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16`
 - `VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16`

- VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16
- VK_FORMAT_A4R4G4B4_UNORM_PACK16
- VK_FORMAT_A4B4G4R4_UNORM_PACK16
- **Packed into 32-bit data types:**
 - VK_FORMAT_A8B8G8R8_UNORM_PACK32
 - VK_FORMAT_A8B8G8R8_SNORM_PACK32
 - VK_FORMAT_A8B8G8R8_USCALED_PACK32
 - VK_FORMAT_A8B8G8R8_SSCALED_PACK32
 - VK_FORMAT_A8B8G8R8_UINT_PACK32
 - VK_FORMAT_A8B8G8R8_SINT_PACK32
 - VK_FORMAT_A8B8G8R8_SRGB_PACK32
 - VK_FORMAT_A2R10G10B10_UNORM_PACK32
 - VK_FORMAT_A2R10G10B10_SNORM_PACK32
 - VK_FORMAT_A2R10G10B10_USCALED_PACK32
 - VK_FORMAT_A2R10G10B10_SSCALED_PACK32
 - VK_FORMAT_A2R10G10B10_UINT_PACK32
 - VK_FORMAT_A2R10G10B10_SINT_PACK32
 - VK_FORMAT_A2B10G10R10_UNORM_PACK32
 - VK_FORMAT_A2B10G10R10_SNORM_PACK32
 - VK_FORMAT_A2B10G10R10_USCALED_PACK32
 - VK_FORMAT_A2B10G10R10_SSCALED_PACK32
 - VK_FORMAT_A2B10G10R10_UINT_PACK32
 - VK_FORMAT_A2B10G10R10_SINT_PACK32
 - VK_FORMAT_B10G11R11_UFLOAT_PACK32
 - VK_FORMAT_E5B9G9R9_UFLOAT_PACK32
 - VK_FORMAT_X8_D24_UNORM_PACK32

35.1.4. Identification of Formats

A “format” is represented by a single enum value. The name of a format is usually built up by using the following pattern:

```
VK_FORMAT_{component-format|compression-scheme}_{numeric-format}
```

The component-format indicates either the size of the R, G, B, and A components (if they are present) in the case of a color format, or the size of the depth (D) and stencil (S) components (if they are present) in the case of a depth/stencil format (see below). An X indicates a component that is

unused, but **may** be present for padding.

Table 51. Interpretation of Numeric Format

Numeric format	Type-Declaration instructions	Numeric type	Description
UNORM	OpTypeFloat	floating-point	The components are unsigned normalized values in the range [0,1]
SNORM	OpTypeFloat	floating-point	The components are signed normalized values in the range [-1,1]
USCALED	OpTypeFloat	floating-point	The components are unsigned integer values that get converted to floating-point in the range [0,2 ⁿ -1]
SSCALED	OpTypeFloat	floating-point	The components are signed integer values that get converted to floating-point in the range [-2 ⁿ⁻¹ ,2 ⁿ⁻¹ -1]
UINT	OpTypeInt	unsigned integer	The components are unsigned integer values in the range [0,2 ⁿ -1]
SINT	OpTypeInt	signed integer	The components are signed integer values in the range [-2 ⁿ⁻¹ ,2 ⁿ⁻¹ -1]
UFLOAT	OpTypeFloat	floating-point	The components are unsigned floating-point numbers (used by packed, shared exponent, and some compressed formats)
SFLOAT	OpTypeFloat	floating-point	The components are signed floating-point numbers
SRGB	OpTypeFloat	floating-point	The R, G, and B components are unsigned normalized values that represent values using sRGB nonlinear encoding, while the A component (if one exists) is a regular unsigned normalized value
n is the number of bits in the component.			

The suffix **_PACKnn** indicates that the format is packed into an underlying type with **nn** bits. The suffix **_mPACKnn** is a short-hand that indicates that the format has **m** groups of components (which may or may not be stored in separate *planes*) that are each packed into an underlying type with **nn** bits.

The suffix **_BLOCK** indicates that the format is a block-compressed format, with the representation of multiple texels encoded interdependently within a region.

Table 52. Interpretation of Compression Scheme

Compression scheme	Description
BC	Block Compression. See Block-Compressed Image Formats .
ETC2	Ericsson Texture Compression. See ETC Compressed Image Formats .
EAC	ETC2 Alpha Compression. See ETC Compressed Image Formats .

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	← Byte		
R		G		VK_FORMAT_R16G16_*														
R		G		B		VK_FORMAT_R16G16B16_*												
R		G		B		A		VK_FORMAT_R16G16B16A16_*										
G ₀		B		G ₁		R		VK_FORMAT_G10X6B10X6G10X6R10X6_4PACK16_422_UNORM										
								VK_FORMAT_G12X4B12X4G12X4R12X4_4PACK16_422_UNORM										
								VK_FORMAT_G16B16G16R16_UNORM										
B		G ₀		R		G ₁		VK_FORMAT_B10X6G10X6R10X6G10X6_4PACK16_422_UNORM										
								VK_FORMAT_B12X4G12X4R12X4G12X4_4PACK16_422_UNORM										
								VK_FORMAT_B16G16R16G16_422_UNORM										
R			VK_FORMAT_R32_*															
R			G				VK_FORMAT_R32G32_*											
R			G				B				VK_FORMAT_R32G32B32_*							
R			G				B				A			VK_FORMAT_R32G32B32A32_*				
R							VK_FORMAT_R64_*											
R							G							VK_FORMAT_R64G64_*				
VK_FORMAT_R64G64B64_* as VK_FORMAT_R64G64_* but with B in bytes 16-23																		
VK_FORMAT_R64G64B64A64_* as VK_FORMAT_R64G64B64_* but with A in bytes 24-31																		

Packed formats store multiple components within one underlying type. The bit representation is that the first component specified in the name of the format is in the most-significant bits and the last component specified is in the least-significant bits of the underlying type. The in-memory ordering of bytes comprising the underlying type is determined by the host endianness.

Table 54. Bit Mappings for Packed 8-Bit Formats

Bit							
7	6	5	4	3	2	1	0
VK_FORMAT_R4G4_UNORM_PACK8							
R				G			
3	2	1	0	3	2	1	0

Table 55. Bit Mappings for Packed 16-Bit Formats

Bit															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VK_FORMAT_R4G4B4A4_UNORM_PACK16															
R				G				B				A			
3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
VK_FORMAT_B4G4R4A4_UNORM_PACK16															
B				G				R				A			
3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0

Bit															
VK_FORMAT_A4R4G4B4_UNORM_PACK16															
A				R				G				B			
3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
VK_FORMAT_A4B4G4R4_UNORM_PACK16															
A				B				G				R			
3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
VK_FORMAT_R5G6B5_UNORM_PACK16															
R					G						B				
4	3	2	1	0	5	4	3	2	1	0	4	3	2	1	0
VK_FORMAT_B5G6R5_UNORM_PACK16															
B					G						R				
4	3	2	1	0	5	4	3	2	1	0	4	3	2	1	0
VK_FORMAT_R5G5B5A1_UNORM_PACK16															
R					G						B				A
4	3	2	1	0	4	3	2	1	0	4	3	2	1	0	0
VK_FORMAT_B5G5R5A1_UNORM_PACK16															
B					G						R				A
4	3	2	1	0	4	3	2	1	0	4	3	2	1	0	0
VK_FORMAT_A1R5G5B5_UNORM_PACK16															
A	R				G						B				
0	4	3	2	1	0	4	3	2	1	0	4	3	2	1	0
VK_FORMAT_R10X6_UNORM_PACK16															
R										X					
9	8	7	6	5	4	3	2	1	0	5	4	3	2	1	0
VK_FORMAT_R12X4_UNORM_PACK16															
R												X			
11	10	9	8	7	6	5	4	3	2	1	0	3	2	1	0

Table 56. Bit Mappings for Packed 32-Bit Formats

Bit																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VK_FORMAT_A8B8G8R8*_PACK32																															
A								B								G								R							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
VK_FORMAT_A2R10G10B10*_PACK32																															
A		R										G										B									
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
VK_FORMAT_A2B10G10R10*_PACK32																															
A		B										G										R									

Bit																															
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
VK_FORMAT_B10G11R11_UFLOAT_PACK32																															
B										G										R											
9	8	7	6	5	4	3	2	1	0	10	9	8	7	6	5	4	3	2	1	0	10	9	8	7	6	5	4	3	2	1	0
VK_FORMAT_E5B9G9R9_UFLOAT_PACK32																															
E					B								G								R										
4	3	2	1	0	8	7	6	5	4	3	2	1	0	8	7	6	5	4	3	2	1	0	8	7	6	5	4	3	2	1	0
VK_FORMAT_X8_D24_UNORM_PACK32																															
X									D																						
7	6	5	4	3	2	1	0	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

35.1.6. Depth/Stencil Formats

Depth/stencil formats are considered opaque and need not be stored in the exact number of bits per texel or component ordering indicated by the format enum. However, implementations **must** not substitute a different depth or stencil precision than is described in the format (e.g. D16 **must** not be implemented as D24 or D32).

35.1.7. Format Compatibility Classes

Uncompressed color formats are *compatible* with each other if they occupy the same number of bits per texel block . Compressed color formats are compatible with each other if the only difference between them is the [numeric format](#) of the uncompressed texels. Each depth/stencil format is only compatible with itself. In the [following](#) table, all the formats in the same row are compatible. Each format has a defined *texel block extent* specifying how many texels each texel block represents in each dimension.

Table 57. Compatible Formats

Class, Texel Block Size, Texel Block Extent, # Texels/Block	Formats
8-bit Block size 1 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R4G4_UNORM_PACK8, VK_FORMAT_R8_UNORM, VK_FORMAT_R8_SNORM, VK_FORMAT_R8_USCALED, VK_FORMAT_R8_SSCALED, VK_FORMAT_R8_UINT, VK_FORMAT_R8_SINT, VK_FORMAT_R8_SRGB

Class, Texel Block Size, Texel Block Extent, # Texels/Block	Formats
16-bit Block size 2 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R10X6_UNORM_PACK16, VK_FORMAT_R12X4_UNORM_PACK16, VK_FORMAT_A4R4G4B4_UNORM_PACK16, VK_FORMAT_A4B4G4R4_UNORM_PACK16, VK_FORMAT_R4G4B4A4_UNORM_PACK16, VK_FORMAT_B4G4R4A4_UNORM_PACK16, VK_FORMAT_R5G6B5_UNORM_PACK16, VK_FORMAT_B5G6R5_UNORM_PACK16, VK_FORMAT_R5G5B5A1_UNORM_PACK16, VK_FORMAT_B5G5R5A1_UNORM_PACK16, VK_FORMAT_A1R5G5B5_UNORM_PACK16, VK_FORMAT_R8G8_UNORM, VK_FORMAT_R8G8_SNORM, VK_FORMAT_R8G8_USCALED, VK_FORMAT_R8G8_SSCALED, VK_FORMAT_R8G8_UINT, VK_FORMAT_R8G8_SINT, VK_FORMAT_R8G8_SRGB, VK_FORMAT_R16_UNORM, VK_FORMAT_R16_SNORM, VK_FORMAT_R16_USCALED, VK_FORMAT_R16_SSCALED, VK_FORMAT_R16_UINT, VK_FORMAT_R16_SINT, VK_FORMAT_R16_SFLOAT
24-bit Block size 3 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R8G8B8_UNORM, VK_FORMAT_R8G8B8_SNORM, VK_FORMAT_R8G8B8_USCALED, VK_FORMAT_R8G8B8_SSCALED, VK_FORMAT_R8G8B8_UINT, VK_FORMAT_R8G8B8_SINT, VK_FORMAT_R8G8B8_SRGB, VK_FORMAT_B8G8R8_UNORM, VK_FORMAT_B8G8R8_SNORM, VK_FORMAT_B8G8R8_USCALED, VK_FORMAT_B8G8R8_SSCALED, VK_FORMAT_B8G8R8_UINT, VK_FORMAT_B8G8R8_SINT, VK_FORMAT_B8G8R8_SRGB

Class, Texel Block Size, Texel Block Extent, # Texels/Block	Formats
32-bit Block size 4 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R10X6G10X6_UNORM_2PACK16, VK_FORMAT_R12X4G12X4_UNORM_2PACK16, VK_FORMAT_R8G8B8A8_UNORM, VK_FORMAT_R8G8B8A8_SNORM, VK_FORMAT_R8G8B8A8_USCALED, VK_FORMAT_R8G8B8A8_SSCALED, VK_FORMAT_R8G8B8A8_UINT, VK_FORMAT_R8G8B8A8_SINT, VK_FORMAT_R8G8B8A8_SRGB, VK_FORMAT_B8G8R8A8_UNORM, VK_FORMAT_B8G8R8A8_SNORM, VK_FORMAT_B8G8R8A8_USCALED, VK_FORMAT_B8G8R8A8_SSCALED, VK_FORMAT_B8G8R8A8_UINT, VK_FORMAT_B8G8R8A8_SINT, VK_FORMAT_B8G8R8A8_SRGB, VK_FORMAT_A8B8G8R8_UNORM_PACK32, VK_FORMAT_A8B8G8R8_SNORM_PACK32, VK_FORMAT_A8B8G8R8_USCALED_PACK32, VK_FORMAT_A8B8G8R8_SSCALED_PACK32, VK_FORMAT_A8B8G8R8_UINT_PACK32, VK_FORMAT_A8B8G8R8_SINT_PACK32, VK_FORMAT_A8B8G8R8_SRGB_PACK32, VK_FORMAT_A2R10G10B10_UNORM_PACK32, VK_FORMAT_A2R10G10B10_SNORM_PACK32, VK_FORMAT_A2R10G10B10_USCALED_PACK32, VK_FORMAT_A2R10G10B10_SSCALED_PACK32, VK_FORMAT_A2R10G10B10_UINT_PACK32, VK_FORMAT_A2R10G10B10_SINT_PACK32, VK_FORMAT_A2B10G10R10_UNORM_PACK32, VK_FORMAT_A2B10G10R10_SNORM_PACK32, VK_FORMAT_A2B10G10R10_USCALED_PACK32, VK_FORMAT_A2B10G10R10_SSCALED_PACK32, VK_FORMAT_A2B10G10R10_UINT_PACK32, VK_FORMAT_A2B10G10R10_SINT_PACK32, VK_FORMAT_R16G16_UNORM, VK_FORMAT_R16G16_SNORM, VK_FORMAT_R16G16_USCALED, VK_FORMAT_R16G16_SSCALED, VK_FORMAT_R16G16_UINT

Class, Texel Block Size, Texel Block Extent, # Texels/Block	Formats
32-bit (continued) Block size 4 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R16G16_SINT, VK_FORMAT_R16G16_SFLOAT, VK_FORMAT_R32_UINT, VK_FORMAT_R32_SINT, VK_FORMAT_R32_SFLOAT, VK_FORMAT_B10G11R11_UFLOAT_PACK32, VK_FORMAT_E5B9G9R9_UFLOAT_PACK32
48-bit Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R16G16B16_UNORM, VK_FORMAT_R16G16B16_SNORM, VK_FORMAT_R16G16B16_USCALED, VK_FORMAT_R16G16B16_SSCALED, VK_FORMAT_R16G16B16_UINT, VK_FORMAT_R16G16B16_SINT, VK_FORMAT_R16G16B16_SFLOAT
64-bit Block size 8 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R16G16B16A16_UNORM, VK_FORMAT_R16G16B16A16_SNORM, VK_FORMAT_R16G16B16A16_USCALED, VK_FORMAT_R16G16B16A16_SSCALED, VK_FORMAT_R16G16B16A16_UINT, VK_FORMAT_R16G16B16A16_SINT, VK_FORMAT_R16G16B16A16_SFLOAT, VK_FORMAT_R32G32_UINT, VK_FORMAT_R32G32_SINT, VK_FORMAT_R32G32_SFLOAT, VK_FORMAT_R64_UINT, VK_FORMAT_R64_SINT, VK_FORMAT_R64_SFLOAT
96-bit Block size 12 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R32G32B32_UINT, VK_FORMAT_R32G32B32_SINT, VK_FORMAT_R32G32B32_SFLOAT
128-bit Block size 16 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R32G32B32A32_UINT, VK_FORMAT_R32G32B32A32_SINT, VK_FORMAT_R32G32B32A32_SFLOAT, VK_FORMAT_R64G64_UINT, VK_FORMAT_R64G64_SINT, VK_FORMAT_R64G64_SFLOAT
192-bit Block size 24 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R64G64B64_UINT, VK_FORMAT_R64G64B64_SINT, VK_FORMAT_R64G64B64_SFLOAT

Class, Texel Block Size, Texel Block Extent, # Texels/Block	Formats
256-bit Block size 32 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R64G64B64A64_UINT, VK_FORMAT_R64G64B64A64_SINT, VK_FORMAT_R64G64B64A64_SFLOAT
D16 Block size 2 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_D16_UNORM
D24 Block size 4 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_X8_D24_UNORM_PACK32
D32 Block size 4 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_D32_SFLOAT
S8 Block size 1 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_S8_UINT
D16S8 Block size 3 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_D16_UNORM_S8_UINT
D24S8 Block size 4 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_D24_UNORM_S8_UINT
D32S8 Block size 5 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_D32_SFLOAT_S8_UINT
BC1_RGB Block size 8 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_BC1_RGB_UNORM_BLOCK, VK_FORMAT_BC1_RGB_SRGB_BLOCK
BC1_RGBA Block size 8 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_BC1_RGBA_UNORM_BLOCK, VK_FORMAT_BC1_RGBA_SRGB_BLOCK

Class, Texel Block Size, Texel Block Extent, # Texels/Block	Formats
BC2 Block size 16 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_BC2_UNORM_BLOCK, VK_FORMAT_BC2_SRGB_BLOCK
BC3 Block size 16 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_BC3_UNORM_BLOCK, VK_FORMAT_BC3_SRGB_BLOCK
BC4 Block size 8 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_BC4_UNORM_BLOCK, VK_FORMAT_BC4_SNORM_BLOCK
BC5 Block size 16 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_BC5_UNORM_BLOCK, VK_FORMAT_BC5_SNORM_BLOCK
BC6H Block size 16 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_BC6H_UFLOAT_BLOCK, VK_FORMAT_BC6H_SFLOAT_BLOCK
BC7 Block size 16 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_BC7_UNORM_BLOCK, VK_FORMAT_BC7_SRGB_BLOCK
ETC2_RGB Block size 8 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK, VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK
ETC2_RGBA Block size 8 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK, VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK
ETC2_EAC_RGBA Block size 16 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK, VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK
EAC_R Block size 8 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_EAC_R11_UNORM_BLOCK, VK_FORMAT_EAC_R11_SNORM_BLOCK

Class, Texel Block Size, Texel Block Extent, # Texels/Block	Formats
EAC_RG Block size 16 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_EAC_R11G11_UNORM_BLOCK, VK_FORMAT_EAC_R11G11_SNORM_BLOCK
ASTC_4x4 Block size 16 byte 4x4x1 block extent 16 texel/block	VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK, VK_FORMAT_ASTC_4x4_UNORM_BLOCK, VK_FORMAT_ASTC_4x4_SRGB_BLOCK
ASTC_5x4 Block size 16 byte 5x4x1 block extent 20 texel/block	VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK, VK_FORMAT_ASTC_5x4_UNORM_BLOCK, VK_FORMAT_ASTC_5x4_SRGB_BLOCK
ASTC_5x5 Block size 16 byte 5x5x1 block extent 25 texel/block	VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK, VK_FORMAT_ASTC_5x5_UNORM_BLOCK, VK_FORMAT_ASTC_5x5_SRGB_BLOCK
ASTC_6x5 Block size 16 byte 6x5x1 block extent 30 texel/block	VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK, VK_FORMAT_ASTC_6x5_UNORM_BLOCK, VK_FORMAT_ASTC_6x5_SRGB_BLOCK
ASTC_6x6 Block size 16 byte 6x6x1 block extent 36 texel/block	VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK, VK_FORMAT_ASTC_6x6_UNORM_BLOCK, VK_FORMAT_ASTC_6x6_SRGB_BLOCK
ASTC_8x5 Block size 16 byte 8x5x1 block extent 40 texel/block	VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK, VK_FORMAT_ASTC_8x5_UNORM_BLOCK, VK_FORMAT_ASTC_8x5_SRGB_BLOCK
ASTC_8x6 Block size 16 byte 8x6x1 block extent 48 texel/block	VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK, VK_FORMAT_ASTC_8x6_UNORM_BLOCK, VK_FORMAT_ASTC_8x6_SRGB_BLOCK
ASTC_8x8 Block size 16 byte 8x8x1 block extent 64 texel/block	VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK, VK_FORMAT_ASTC_8x8_UNORM_BLOCK, VK_FORMAT_ASTC_8x8_SRGB_BLOCK
ASTC_10x5 Block size 16 byte 10x5x1 block extent 50 texel/block	VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK, VK_FORMAT_ASTC_10x5_UNORM_BLOCK, VK_FORMAT_ASTC_10x5_SRGB_BLOCK

Class, Texel Block Size, Texel Block Extent, # Texels/Block	Formats
ASTC_10x6 Block size 16 byte 10x6x1 block extent 60 texel/block	VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK, VK_FORMAT_ASTC_10x6_UNORM_BLOCK, VK_FORMAT_ASTC_10x6_SRGB_BLOCK
ASTC_10x8 Block size 16 byte 10x8x1 block extent 80 texel/block	VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK, VK_FORMAT_ASTC_10x8_UNORM_BLOCK, VK_FORMAT_ASTC_10x8_SRGB_BLOCK
ASTC_10x10 Block size 16 byte 10x10x1 block extent 100 texel/block	VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK, VK_FORMAT_ASTC_10x10_UNORM_BLOCK, VK_FORMAT_ASTC_10x10_SRGB_BLOCK
ASTC_12x10 Block size 16 byte 12x10x1 block extent 120 texel/block	VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK, VK_FORMAT_ASTC_12x10_UNORM_BLOCK, VK_FORMAT_ASTC_12x10_SRGB_BLOCK
ASTC_12x12 Block size 16 byte 12x12x1 block extent 144 texel/block	VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK, VK_FORMAT_ASTC_12x12_UNORM_BLOCK, VK_FORMAT_ASTC_12x12_SRGB_BLOCK
32-bit G8B8G8R8 Block size 4 byte 2x1x1 block extent 1 texel/block	VK_FORMAT_G8B8G8R8_422_UNORM
32-bit B8G8R8G8 Block size 4 byte 2x1x1 block extent 1 texel/block	VK_FORMAT_B8G8R8G8_422_UNORM
8-bit 3-plane 420 Block size 3 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM
8-bit 2-plane 420 Block size 3 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G8_B8R8_2PLANE_420_UNORM
8-bit 3-plane 422 Block size 3 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM

Class, Texel Block Size, Texel Block Extent, # Texels/Block	Formats
8-bit 2-plane 422 Block size 3 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G8_B8R8_2PLANE_422_UNORM
8-bit 3-plane 444 Block size 3 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM
64-bit R10G10B10A10 Block size 8 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16,
64-bit G10B10G10R10 Block size 8 byte 2x1x1 block extent 1 texel/block	VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16
64-bit B10G10R10G10 Block size 8 byte 2x1x1 block extent 1 texel/block	VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16
10-bit 3-plane 420 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16
10-bit 2-plane 420 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16
10-bit 3-plane 422 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16
10-bit 2-plane 422 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16
10-bit 3-plane 444 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16

Class, Texel Block Size, Texel Block Extent, # Texels/Block	Formats
64-bit R12G12B12A12 Block size 8 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16,
64-bit G12B12G12R12 Block size 8 byte 2x1x1 block extent 1 texel/block	VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16
64-bit B12G12R12G12 Block size 8 byte 2x1x1 block extent 1 texel/block	VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16
12-bit 3-plane 420 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16
12-bit 2-plane 420 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16
12-bit 3-plane 422 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16
12-bit 2-plane 422 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16
12-bit 3-plane 444 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16
64-bit G16B16G16R16 Block size 8 byte 2x1x1 block extent 1 texel/block	VK_FORMAT_G16B16G16R16_422_UNORM
64-bit B16G16R16G16 Block size 8 byte 2x1x1 block extent 1 texel/block	VK_FORMAT_B16G16R16G16_422_UNORM

Class, Texel Block Size, Texel Block Extent, # Texels/Block	Formats
16-bit 3-plane 420 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM
16-bit 2-plane 420 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G16_B16R16_2PLANE_420_UNORM
16-bit 3-plane 422 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM
16-bit 2-plane 422 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G16_B16R16_2PLANE_422_UNORM
16-bit 3-plane 444 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM
8-bit 2-plane 444 Block size 3 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G8_B8R8_2PLANE_444_UNORM
10-bit 2-plane 444 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16
12-bit 2-plane 444 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16
16-bit 2-plane 444 Block size 6 byte 1x1x1 block extent 1 texel/block	VK_FORMAT_G16_B16R16_2PLANE_444_UNORM

Size Compatibility

Color formats with the same texel block size are considered *size-compatible*. If two size-compatible formats have different block extents (i.e. for compressed formats), then an image with size $A \times B \times C$

in one format with a block extent of $a \times b \times c$ can be represented as an image with size $X \times Y \times Z$ in the other format with block extent $x \times y \times z$ at the ratio between the block extents for each format, where

$$\lceil A/a \rceil = \lceil X/x \rceil$$

$$\lceil B/b \rceil = \lceil Y/y \rceil$$

$$\lceil C/c \rceil = \lceil Z/z \rceil$$

NOTE

For example, a 7x3 image in the `VK_FORMAT_ASTC_8x5_UNORM_BLOCK` format can be represented as a 1x1 `VK_FORMAT_R64G64_UINT` image.

Images created with the `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT` flag can have size-compatible views created from them to enable access via different size-compatible formats. Image views created in this way will be sized to match the expectations of the block extents noted above.

Copy operations are able to copy between size-compatible formats in different resources to enable manipulation of data in different formats. The extent used in these copy operations always matches the source image, and is resized to the expectations of the block extents noted above for the destination image.

35.2. Format Properties

To query supported format features which are properties of the physical device, call:

WARNING

This functionality is superseded by [Vulkan Version 1.1](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
void vkGetPhysicalDeviceFormatProperties(
    VkPhysicalDevice    physicalDevice,
    VkFormat             format,
    VkFormatProperties*  pFormatProperties);
```

- `physicalDevice` is the physical device from which to query the format properties.
- `format` is the format whose properties are queried.
- `pFormatProperties` is a pointer to a `VkFormatProperties` structure in which physical device properties for `format` are returned.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceFormatProperties-physicalDevice-parameter

`physicalDevice` **must** be a valid `VkPhysicalDevice` handle

- `vkGetPhysicalDeviceFormatProperties-format-parameter`
`format` **must** be a valid `VkFormat` value
- `vkGetPhysicalDeviceFormatProperties-pFormatProperties-parameter`
`pFormatProperties` **must** be a valid pointer to a `VkFormatProperties` structure

The `VkFormatProperties` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkFormatProperties {
    VkFormatFeatureFlags    linearTilingFeatures;
    VkFormatFeatureFlags    optimalTilingFeatures;
    VkFormatFeatureFlags    bufferFeatures;
} VkFormatProperties;
```

- `linearTilingFeatures` is a bitmask of `VkFormatFeatureFlagBits` specifying features supported by images created with a `tiling` parameter of `VK_IMAGE_TILING_LINEAR`.
- `optimalTilingFeatures` is a bitmask of `VkFormatFeatureFlagBits` specifying features supported by images created with a `tiling` parameter of `VK_IMAGE_TILING_OPTIMAL`.
- `bufferFeatures` is a bitmask of `VkFormatFeatureFlagBits` specifying features supported by buffers.

NOTE

If no format feature flags are supported, the format itself is not supported, and images of that format cannot be created.

If `format` is block-compressed, [requires sampler Y'C_BC_R conversion](#), or is a depth/stencil format then `bufferFeatures` **must** not support any features for the format.

If `format` is not a multi-plane format then `linearTilingFeatures` and `optimalTilingFeatures` **must** not contain `VK_FORMAT_FEATURE_DISJOINT_BIT`.

Bits which **can** be set in the `VkFormatProperties` features `linearTilingFeatures`, `optimalTilingFeatures`, `VkDrmFormatModifierPropertiesEXT::drmFormatModifierTilingFeatures`, and `bufferFeatures` are:

```

// Provided by VK_VERSION_1_0
typedef enum VkFormatFeatureFlagBits {
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT = 0x00000001,
    VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT = 0x00000002,
    VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT = 0x00000004,
    VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT = 0x00000008,
    VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT = 0x00000010,
    VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT = 0x00000020,
    VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT = 0x00000040,
    VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT = 0x00000080,
    VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT = 0x00000100,
    VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT = 0x00000200,
    VK_FORMAT_FEATURE_BLIT_SRC_BIT = 0x00000400,
    VK_FORMAT_FEATURE_BLIT_DST_BIT = 0x00000800,
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT = 0x00001000,
// Provided by VK_VERSION_1_1
    VK_FORMAT_FEATURE_TRANSFER_SRC_BIT = 0x00004000,
// Provided by VK_VERSION_1_1
    VK_FORMAT_FEATURE_TRANSFER_DST_BIT = 0x00008000,
// Provided by VK_VERSION_1_1
    VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT = 0x00020000,
// Provided by VK_VERSION_1_1
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT = 0x00040000,
// Provided by VK_VERSION_1_1

VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT =
0x00080000,
    // Provided by VK_VERSION_1_1

VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT =
0x00100000,
    // Provided by VK_VERSION_1_1

VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT =
0x00200000,
    // Provided by VK_VERSION_1_1
    VK_FORMAT_FEATURE_DISJOINT_BIT = 0x00400000,
// Provided by VK_VERSION_1_1
    VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT = 0x00800000,
// Provided by VK_VERSION_1_2
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT = 0x00010000,
// Provided by VK_EXT_filter_cubic
    VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT = 0x00002000,
// Provided by VK_KHR_fragment_shading_rate
    VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR = 0x40000000,
} VkFormatFeatureFlagBits;

```

These values **may** be set in `linearTilingFeatures`, `optimalTilingFeatures`, and `VkDrmFormatModifierPropertiesEXT::drmFormatModifierTilingFeatures`, specifying that the features

are supported by [images](#) or [image views](#) or [sampler Y'C_BC_R conversion objects](#) created with the queried `vkGetPhysicalDeviceFormatProperties::format:`

- `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT` specifies that an image view **can** be [sampled from](#).
- `VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT` specifies that an image view **can** be used as a [storage image](#).
- `VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT` specifies that an image view **can** be used as storage image that supports atomic operations.
- `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` specifies that an image view **can** be used as a framebuffer color attachment and as an input attachment.
- `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT` specifies that an image view **can** be used as a framebuffer color attachment that supports blending.
- `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT` specifies that an image view **can** be used as a framebuffer depth/stencil attachment and as an input attachment.
- `VK_FORMAT_FEATURE_BLIT_SRC_BIT` specifies that an image **can** be used as `srcImage` for the `vkCmdBlitImage2KHR` and `vkCmdBlitImage` commands.
- `VK_FORMAT_FEATURE_BLIT_DST_BIT` specifies that an image **can** be used as `dstImage` for the `vkCmdBlitImage2KHR` and `vkCmdBlitImage` commands.
- `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` specifies that if `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT` is also set, an image view **can** be used with a sampler that has either of `magFilter` or `minFilter` set to `VK_FILTER_LINEAR`, or `mipmapMode` set to `VK_SAMPLER_MIPMAP_MODE_LINEAR`. If `VK_FORMAT_FEATURE_BLIT_SRC_BIT` is also set, an image can be used as the `srcImage` to `vkCmdBlitImage2KHR` and `vkCmdBlitImage` with a `filter` of `VK_FILTER_LINEAR`. This bit **must** only be exposed for formats that also support the `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT` or `VK_FORMAT_FEATURE_BLIT_SRC_BIT`.

If the format being queried is a depth/stencil format, this bit only specifies that the depth aspect (not the stencil aspect) of an image of this format supports linear filtering, and that linear filtering of the depth aspect is supported whether depth compare is enabled in the sampler or not. Where depth comparison is supported it **may** be linear filtered whether this bit is present or not, but where this bit is not present the filtered value **may** be computed in an implementation-dependent manner which differs from the normal rules of linear filtering. The resulting value **must** be in the range [0,1] and **should** be proportional to, or a weighted average of, the number of comparison passes or failures.

- `VK_FORMAT_FEATURE_TRANSFER_SRC_BIT` specifies that an image **can** be used as a source image for [copy commands](#). If the application `apiVersion` is Vulkan 1.0 and `VK_KHR_maintenance1` is not supported, `VK_FORMAT_FEATURE_TRANSFER_SRC_BIT` is implied to be set when the format feature flag is not 0.
- `VK_FORMAT_FEATURE_TRANSFER_DST_BIT` specifies that an image **can** be used as a destination image for [copy commands](#) and [clear commands](#). If the application `apiVersion` is Vulkan 1.0 and `VK_KHR_maintenance1` is not supported, `VK_FORMAT_FEATURE_TRANSFER_DST_BIT` is implied to be set when the format feature flag is not 0.
- `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT` specifies `VkImage` **can** be used as a sampled

image with a min or max [VkSamplerReductionMode](#). This bit **must** only be exposed for formats that also support the [VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT](#).

- [VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT](#) specifies that [VkImage](#) **can** be used with a sampler that has either of [magFilter](#) or [minFilter](#) set to [VK_FILTER_CUBIC_EXT](#), or be the source image for a blit with [filter](#) set to [VK_FILTER_CUBIC_EXT](#). This bit **must** only be exposed for formats that also support the [VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT](#). If the format being queried is a depth/stencil format, this only specifies that the depth aspect is cubic filterable.
- [VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT](#) specifies that an application **can** define a [sampler Y'C_BC_R conversion](#) using this format as a source, and that an image of this format **can** be used with a [VkSamplerYcbcrConversionCreateInfo](#) [xChromaOffset](#) and/or [yChromaOffset](#) of [VK_CHROMA_LOCATION_MIDPOINT](#). Otherwise both [xChromaOffset](#) and [yChromaOffset](#) **must** be [VK_CHROMA_LOCATION_COSITED_EVEN](#). If a format does not incorporate chroma downsampling (it is not a “422” or “420” format) but the implementation supports sampler Y'C_BC_R conversion for this format, the implementation **must** set [VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT](#).
- [VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT](#) specifies that an application **can** define a [sampler Y'C_BC_R conversion](#) using this format as a source, and that an image of this format **can** be used with a [VkSamplerYcbcrConversionCreateInfo](#) [xChromaOffset](#) and/or [yChromaOffset](#) of [VK_CHROMA_LOCATION_COSITED_EVEN](#). Otherwise both [xChromaOffset](#) and [yChromaOffset](#) **must** be [VK_CHROMA_LOCATION_MIDPOINT](#). If neither [VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT](#) nor [VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT](#) is set, the application **must** not define a [sampler Y'C_BC_R conversion](#) using this format as a source.
- [VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT](#) specifies that an application **can** define a [sampler Y'C_BC_R conversion](#) using this format as a source with [chromaFilter](#) set to [VK_FILTER_LINEAR](#).
- [VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT](#) specifies that the format can have different chroma, min, and mag filters.
- [VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT](#) specifies that reconstruction is explicit, as described in [Chroma Reconstruction](#). If this bit is not present, reconstruction is implicit by default.
- [VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT](#) specifies that reconstruction **can** be forcibly made explicit by setting [VkSamplerYcbcrConversionCreateInfo::forceExplicitReconstruction](#) to [VK_TRUE](#). If the format being queried supports [VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT](#) it **must** also support [VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT](#).
- [VK_FORMAT_FEATURE_DISJOINT_BIT](#) specifies that a multi-planar image **can** have the [VK_IMAGE_CREATE_DISJOINT_BIT](#) set during image creation. An implementation **must** not set [VK_FORMAT_FEATURE_DISJOINT_BIT](#) for *single-plane formats*.
- [VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR](#) specifies that an image view **can** be used as a [fragment shading rate attachment](#). An implementation **must** not set this feature for formats with a [numeric format](#) other than [UINT](#), or set it as a buffer feature.

The following bits **may** be set in `bufferFeatures`, specifying that the features are supported by `buffers` or `buffer views` created with the queried `vkGetPhysicalDeviceFormatProperties::format`:

- `VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT` specifies that the format **can** be used to create a buffer view that **can** be bound to a `VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER` descriptor.
- `VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT` specifies that the format **can** be used to create a buffer view that **can** be bound to a `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` descriptor.
- `VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT` specifies that atomic operations are supported on `VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER` with this format.
- `VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT` specifies that the format **can** be used as a vertex attribute format (`VkVertexInputAttributeDescription::format`).

NOTE

`VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT` and `VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT` are only intended to be advertised for single-component formats, since SPIR-V atomic operations require a scalar type.

```
// Provided by VK_VERSION_1_0
typedef VkFlags VkFormatFeatureFlags;
```

`VkFormatFeatureFlags` is a bitmask type for setting a mask of zero or more `VkFormatFeatureFlagBits`.

To query supported format features which are properties of the physical device, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceFormatProperties2(
    VkPhysicalDevice          physicalDevice,
    VkFormat                  format,
    VkFormatProperties2*      pFormatProperties);
```

- `physicalDevice` is the physical device from which to query the format properties.
- `format` is the format whose properties are queried.
- `pFormatProperties` is a pointer to a `VkFormatProperties2` structure in which physical device properties for `format` are returned.

`vkGetPhysicalDeviceFormatProperties2` behaves similarly to `vkGetPhysicalDeviceFormatProperties`, with the ability to return extended information in a `pNext` chain of output structures.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceFormatProperties2-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceFormatProperties2-format-parameter `format` **must** be a valid `VkFormat` value

- VUID-vkGetPhysicalDeviceFormatProperties2-pFormatProperties-parameter **pFormatProperties** **must** be a valid pointer to a [VkFormatProperties2](#) structure

The [VkFormatProperties2](#) structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkFormatProperties2 {
    VkStructureType    sType;
    void*              pNext;
    VkFormatProperties  formatProperties;
} VkFormatProperties2;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **formatProperties** is a [VkFormatProperties](#) structure describing features supported by the requested format.

Valid Usage (Implicit)

- VUID-VkFormatProperties2-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_2**
- VUID-VkFormatProperties2-pNext-pNext
Each **pNext** member of any structure (including this one) in the **pNext** chain **must** be either **NULL** or a pointer to a valid instance of [VkDrmFormatModifierPropertiesList2EXT](#) or [VkDrmFormatModifierPropertiesListEXT](#)
- VUID-VkFormatProperties2-sType-unique
The **sType** value of each structure in the **pNext** chain **must** be unique

To obtain the list of [Linux DRM format modifiers](#) compatible with a [VkFormat](#), add a [VkDrmFormatModifierPropertiesListEXT](#) structure to the **pNext** chain of [VkFormatProperties2](#).

The [VkDrmFormatModifierPropertiesListEXT](#) structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier
typedef struct VkDrmFormatModifierPropertiesListEXT {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           drmFormatModifierCount;
    VkDrmFormatModifierPropertiesEXT* pDrmFormatModifierProperties;
} VkDrmFormatModifierPropertiesListEXT;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.

- `drmFormatModifierCount` is an inout parameter related to the number of modifiers compatible with the `format`, as described below.
- `pDrmFormatModifierProperties` is either `NULL` or a pointer to an array of `VkDrmFormatModifierPropertiesEXT` structures.

If `pDrmFormatModifierProperties` is `NULL`, then the function returns in `drmFormatModifierCount` the number of modifiers compatible with the queried `format`. Otherwise, the application **must** set `drmFormatModifierCount` to the length of the array `pDrmFormatModifierProperties`; the function will write at most `drmFormatModifierCount` elements to the array, and will return in `drmFormatModifierCount` the number of elements written.

Among the elements in array `pDrmFormatModifierProperties`, each returned `drmFormatModifier` **must** be unique.

Valid Usage (Implicit)

- `VUID-VkDrmFormatModifierPropertiesListEXT-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_DRM_FORMAT_MODIFIER_PROPERTIES_LIST_EXT`

The `VkDrmFormatModifierPropertiesEXT` structure describes properties of a `VkFormat` when that format is combined with a `Linux DRM format modifier`. These properties, like those of `VkFormatProperties2`, are independent of any particular image.

The `VkDrmFormatModifierPropertiesEXT` structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier
typedef struct VkDrmFormatModifierPropertiesEXT {
    uint64_t          drmFormatModifier;
    uint32_t          drmFormatModifierPlaneCount;
    VkFormatFeatureFlags  drmFormatModifierTilingFeatures;
} VkDrmFormatModifierPropertiesEXT;
```

- `drmFormatModifier` is a *Linux DRM format modifier*.
- `drmFormatModifierPlaneCount` is the number of *memory planes* in any image created with `format` and `drmFormatModifier`. An image's *memory planecount* is distinct from its *format planecount*, as explained below.
- `drmFormatModifierTilingFeatures` is a bitmask of `VkFormatFeatureFlagBits` that are supported by any image created with `format` and `drmFormatModifier`.

The returned `drmFormatModifierTilingFeatures` **must** contain at least one bit.

The implementation **must** not return `DRM_FORMAT_MOD_INVALID` in `drmFormatModifier`.

An image's *memory planecount* (as returned by `drmFormatModifierPlaneCount`) is distinct from its *format planecount* (in the sense of `multi-planar format`). In `VkImageAspectFlags`, each `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT` represents a *memory plane* and each

`VK_IMAGE_ASPECT_PLANE_i_BIT` a *format plane*.

An image's set of *format planes* is an ordered partition of the image's **content** into separable groups of format components. The ordered partition is encoded in the name of each `VkFormat`. For example, `VK_FORMAT_G8_B8R8_2PLANE_420_UNORM` contains two *format planes*; the first plane contains the green component and the second plane contains the blue component and red component. If the format name does not contain `PLANE`, then the format contains a single plane; for example, `VK_FORMAT_R8G8B8A8_UNORM`. Some commands, such as `vkCmdCopyBufferToImage`, do not operate on all format components in the image, but instead operate only on the *format planes* explicitly chosen by the application and operate on each *format plane* independently.

An image's set of *memory planes* is an ordered partition of the image's **memory** rather than the image's **content**. Each *memory plane* is a contiguous range of memory. The union of an image's *memory planes* is not necessarily contiguous.

If an image is *linear*, then the partition is the same for *memory planes* and for *format planes*. Therefore, if the returned `drmFormatModifier` is `DRM_FORMAT_MOD_LINEAR`, then `drmFormatModifierPlaneCount` **must** equal the *format plane*count, and `drmFormatModifierTilingFeatures` **must** be identical to the `VkFormatProperties2::formatProperties.linearTilingFeatures` returned in the same `pNext` chain.

If an image is *non-linear*, then the partition of the image's **memory** into *memory planes* is implementation-specific and **may** be unrelated to the partition of the image's **content** into *format planes*. For example, consider an image whose *format* is `VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM`, *tiling* is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, whose `drmFormatModifier` is not `DRM_FORMAT_MOD_LINEAR`, and `flags` lacks `VK_IMAGE_CREATE_DISJOINT_BIT`. The image has 3 *format planes*, and commands such `vkCmdCopyBufferToImage` act on each *format plane* independently as if the data of each *format plane* were separable from the data of the other planes. In a straightforward implementation, the implementation **may** store the image's content in 3 adjacent *memory planes* where each *memory plane* corresponds exactly to a *format plane*. However, the implementation **may** also store the image's content in a single *memory plane* where all format components are combined using an implementation-private block-compressed format; or the implementation **may** store the image's content in a collection of 7 adjacent *memory planes* using an implementation-private sharding technique. Because the image is non-linear and non-disjoint, the implementation has much freedom when choosing the image's placement in memory.

The *memory plane*count applies to function parameters and structures only when the API specifies an explicit requirement on `drmFormatModifierPlaneCount`. In all other cases, the *memory plane*count is ignored.

35.2.1. Potential Format Features

Some *valid usage conditions* depend on the format features supported by a `VkImage` whose `VkImageTiling` is unknown. In such cases the exact `VkFormatFeatureFlagBits` supported by the `VkImage` cannot be determined, so the valid usage conditions are expressed in terms of the *potential format features* of the `VkImage` format.

The *potential format features* of a `VkFormat` are defined as follows:

- The union of `VkFormatFeatureFlagBits` supported when the `VkImageTiling` is `VK_IMAGE_TILING_OPTIMAL`, `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, or `VK_IMAGE_TILING_LINEAR` if `VkFormat` is not `VK_FORMAT_UNDEFINED`
- `VkScreenBufferFormatPropertiesQNX::formatFeatures` of a valid external format if `VkFormat` is `VK_FORMAT_UNDEFINED`

35.3. Required Format Support

Implementations **must** support at least the following set of features on the listed formats. For images, these features **must** be supported for every `VkImageType` (including arrayed and cube variants) unless otherwise noted. These features are supported on existing formats without needing to advertise an extension or needing to explicitly enable them. Support for additional functionality beyond the requirements listed here is queried using the `vkGetPhysicalDeviceFormatProperties` command.

NOTE

Unless otherwise excluded below, the required formats are supported for all `VkImageCreateFlags` values as long as those flag values are otherwise allowed.

The following tables show which feature bits **must** be supported for each format. Formats that are required to support `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT` **must** also support `VK_FORMAT_FEATURE_TRANSFER_SRC_BIT` and `VK_FORMAT_FEATURE_TRANSFER_DST_BIT`.

Table 58. Key for Format Feature Tables

□	This feature must be supported on the named format
†	This feature must be supported on at least some of the named formats, with more information in the table where the symbol appears
‡	This feature must be supported with some caveats or preconditions, with more information in the table where the symbol appears
§	This feature must be supported with some caveats or preconditions, with more information in the table where the symbol appears

Table 59. Feature Bits in `optimalTilingFeatures`

<code>VK_FORMAT_FEATURE_TRANSFER_SRC_BIT</code>
<code>VK_FORMAT_FEATURE_TRANSFER_DST_BIT</code>
<code>VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT</code>
<code>VK_FORMAT_FEATURE_BLIT_SRC_BIT</code>
<code>VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT</code>
<code>VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT</code>
<code>VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT</code>
<code>VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT</code>
<code>VK_FORMAT_FEATURE_BLIT_DST_BIT</code>

VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT
VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT

Table 60. Feature Bits in `bufferFeatures`

VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT
VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT
VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT
VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT

Table 61. Mandatory Format Support: Sub-Byte Components

	VK_FORMAT_FEATURE_2_COPY_IMAGE_INDIRECT_DST_BIT_KHR													
	VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT													
	VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT													
	VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT													
	VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT													
	VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT													
	VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT													
	VK_FORMAT_FEATURE_BLIT_DST_BIT													
	VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT													
	VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT													
	VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT													
	VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT													
	VK_FORMAT_FEATURE_BLIT_SRC_BIT													
	VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT													
Format														
VK_FORMAT_UNDEFINED														
VK_FORMAT_R4G4_UNORM_PACK8														
VK_FORMAT_R4G4B4A4_UNORM_PACK16														
VK_FORMAT_B4G4R4A4_UNORM_PACK16	‡	‡	‡											
VK_FORMAT_R5G6B5_UNORM_PACK16	‡	‡	‡			‡	‡	‡						
VK_FORMAT_B5G6R5_UNORM_PACK16														
VK_FORMAT_R5G5B5A1_UNORM_PACK16														
VK_FORMAT_B5G5R5A1_UNORM_PACK16														
VK_FORMAT_A1R5G5B5_UNORM_PACK16	‡	‡	‡			‡	‡	‡						
VK_FORMAT_A4R4G4B4_UNORM_PACK16	†	†	†											
VK_FORMAT_A4B4G4R4_UNORM_PACK16	‡	‡	‡											
Format features marked † must be supported for <code>optimalTilingFeatures</code> if the <code>VkPhysicalDevice</code> supports the <code>VkPhysicalDevice4444FormatsFeaturesEXT::formatA4R4G4B4</code> feature.														
Format features marked ‡ must be supported for <code>optimalTilingFeatures</code> if the <code>VkPhysicalDevice</code> supports the <code>VkPhysicalDevice4444FormatsFeaturesEXT::formatA4B4G4R4</code> feature.														

Table 62. Mandatory Format Support: 1-3 Byte-Sized Components

[illegible]

VK_FORMAT_B8G8R8_SNORM																			
VK_FORMAT_B8G8R8_USCALED																			
VK_FORMAT_B8G8R8_SSCALED																			
VK_FORMAT_B8G8R8_UINT																			
VK_FORMAT_B8G8R8_SINT																			
VK_FORMAT_B8G8R8_SRGB																			
Format features marked with ‡ must be supported for optimalTilingFeatures if the VkPhysicalDevice supports the shaderStorageImageExtendedFormats feature.																			

Table 63. Mandatory Format Support: 4 Byte-Sized Components

[illegible]

Table 64. Mandatory Format Support: 10-, 12-, and 14-Bit Components

[illegible]

Table 65. Mandatory Format Support: 16-bit Components

VK_FORMAT_FEATURE_2_COPY_IMAGE_INDIRECT_DST_BIT_KHR													
VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT													
VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT													
VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT													
VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT													
VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT													
VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT													
VK_FORMAT_FEATURE_BLIT_DST_BIT													
VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT													
VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT													
VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT													
VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT													
VK_FORMAT_FEATURE_BLIT_SRC_BIT													
VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT													
Format													
VK_FORMAT_R16_UNORM				‡						□			□
VK_FORMAT_R16_SNORM				‡						□			□
VK_FORMAT_R16_USCALED													
VK_FORMAT_R16_SSCALED													
VK_FORMAT_R16_UINT	□	□		‡		□	□			□	□	□	
VK_FORMAT_R16_SINT	□	□		‡		□	□			□	□	□	
VK_FORMAT_R16_SFLOAT	□	□	□	‡		□	□	□		□	□	□	
VK_FORMAT_R16G16_UNORM				‡						□		□	
VK_FORMAT_R16G16_SNORM				‡						□		□	
VK_FORMAT_R16G16_USCALED													
VK_FORMAT_R16G16_SSCALED													
VK_FORMAT_R16G16_UINT	□	□		‡		□	□			□	□	□	
VK_FORMAT_R16G16_SINT	□	□		‡		□	□			□	□	□	
VK_FORMAT_R16G16_SFLOAT	□	□	□	‡	§	□	□	□		□	□	§	
VK_FORMAT_R16G16B16_UNORM													
VK_FORMAT_R16G16B16_SNORM													
VK_FORMAT_R16G16B16_USCALED													
VK_FORMAT_R16G16B16_SSCALED													
VK_FORMAT_R16G16B16_UINT													
VK_FORMAT_R16G16B16_SINT													
VK_FORMAT_R16G16B16_SFLOAT													
VK_FORMAT_R16G16B16A16_UNORM				‡						□		□	

Table 66. Mandatory Format Support: 32-bit Components

	VK_FORMAT_FEATURE_2_COPY_IMAGE_INDIRECT_DST_BIT_KHR													
	VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT													
	VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT													
	VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT													
	VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT													
	VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT													
	VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT													
	VK_FORMAT_FEATURE_BLIT_DST_BIT													
	VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT													
	VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT													
	VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT													
	VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT													
	VK_FORMAT_FEATURE_BLIT_SRC_BIT													
	VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT													
Format														
VK_FORMAT_R32_UINT														
VK_FORMAT_R32_SINT														
VK_FORMAT_R32_SFLOAT					†									
VK_FORMAT_R32G32_UINT														
VK_FORMAT_R32G32_SINT														
VK_FORMAT_R32G32_SFLOAT														
VK_FORMAT_R32G32B32_UINT														
VK_FORMAT_R32G32B32_SINT														
VK_FORMAT_R32G32B32_SFLOAT														
VK_FORMAT_R32G32B32A32_UINT														
VK_FORMAT_R32G32B32A32_SINT														
VK_FORMAT_R32G32B32A32_SFLOAT														
Format features marked with † must be supported for optimalTilingFeatures if the VkPhysicalDevice supports the shaderImageFloat32Atomics or the shaderImageFloat32AtomicAdd feature.														

Table 67. Mandatory Format Support: 64-bit/uneven Components

	VK_FORMAT_FEATURE_2_COPY_IMAGE_INDIRECT_DST_BIT_KHR													
	VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT													
	VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT													
	VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT													
	VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT													
	VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT													
	VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT													
	VK_FORMAT_FEATURE_BLIT_DST_BIT													
	VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT													
	VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT													
	VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT													
	VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT													
	VK_FORMAT_FEATURE_BLIT_SRC_BIT													
	VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT													
Format														
VK_FORMAT_R64_UINT					†	†								
VK_FORMAT_R64_SINT					†	†								
VK_FORMAT_R64_SFLOAT														
VK_FORMAT_R64G64_UINT														
VK_FORMAT_R64G64_SINT														
VK_FORMAT_R64G64_SFLOAT														
VK_FORMAT_R64G64B64_UINT														
VK_FORMAT_R64G64B64_SINT														
VK_FORMAT_R64G64B64_SFLOAT														
VK_FORMAT_R64G64B64A64_UINT														
VK_FORMAT_R64G64B64A64_SINT														
VK_FORMAT_R64G64B64A64_SFLOAT														
VK_FORMAT_B10G11R11_UFLOAT_PACK32	☐	☐	☐	‡							☐		☐	
VK_FORMAT_E5B9G9R9_UFLOAT_PACK32	☐	☐	☐											
Format features marked with ‡ must be supported for optimalTilingFeatures if the VkPhysicalDevice supports the shaderStorageImageExtendedFormats feature.														
If the shaderImageInt64Atomics feature is supported, VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT and VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT must be advertised in optimalTilingFeatures for both VK_FORMAT_R64_UINT and VK_FORMAT_R64_SINT.														

Table 68. Mandatory Format Support: Depth/Stencil With `VkImageType VK_IMAGE_TYPE_2D`

	VK_FORMAT_FEATURE_2_COPY_IMAGE_INDIRECT_DST_BIT_KHR												
	VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT												
	VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT												
	VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT												
	VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT												
	VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT												
	VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT												
	VK_FORMAT_FEATURE_BLIT_DST_BIT												
	VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT												
	VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT												
	VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT												
	VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT												
	VK_FORMAT_FEATURE_BLIT_SRC_BIT												
	VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT												
Format													
VK_FORMAT_D16_UNORM													
VK_FORMAT_X8_D24_UNORM_PACK32													
VK_FORMAT_D32_SFLOAT													
VK_FORMAT_S8_UINT													
VK_FORMAT_D16_UNORM_S8_UINT													
VK_FORMAT_D24_UNORM_S8_UINT													
VK_FORMAT_D32_SFLOAT_S8_UINT													
VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT feature must be supported for at least one of VK_FORMAT_X8_D24_UNORM_PACK32 and VK_FORMAT_D32_SFLOAT, and must be supported for at least one of VK_FORMAT_D24_UNORM_S8_UINT and VK_FORMAT_D32_SFLOAT_S8_UINT.													
bufferFeatures must not support any features for these formats													

Table 69. Mandatory Format Support: BC Compressed Formats With `VkImageType VK_IMAGE_TYPE_2D` and `VK_IMAGE_TYPE_3D`

VK_FORMAT_FEATURE_2_COPY_IMAGE_INDIRECT_DST_BIT_KHR													
VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT													
VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT													
VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT													
VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT													
VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT													
VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT													
VK_FORMAT_FEATURE_BLIT_DST_BIT													
VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT													
VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT													
VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT													
VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT													
VK_FORMAT_FEATURE_BLIT_SRC_BIT													
VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT													
Format	1	2	3	4	5	6	7	8	9	10	11	12	
VK_FORMAT_BC1_RGB_UNORM_BLOCK	†	†	†										
VK_FORMAT_BC1_RGB_SRGB_BLOCK	†	†	†										
VK_FORMAT_BC1_RGBA_UNORM_BLOCK	†	†	†										
VK_FORMAT_BC1_RGBA_SRGB_BLOCK	†	†	†										
VK_FORMAT_BC2_UNORM_BLOCK	†	†	†										
VK_FORMAT_BC2_SRGB_BLOCK	†	†	†										
VK_FORMAT_BC3_UNORM_BLOCK	†	†	†										
VK_FORMAT_BC3_SRGB_BLOCK	†	†	†										
VK_FORMAT_BC4_UNORM_BLOCK	†	†	†										
VK_FORMAT_BC4_SNORM_BLOCK	†	†	†										
VK_FORMAT_BC5_UNORM_BLOCK	†	†	†										
VK_FORMAT_BC5_SNORM_BLOCK	†	†	†										
VK_FORMAT_BC6H_UFLOAT_BLOCK	†	†	†										
VK_FORMAT_BC6H_SFLOAT_BLOCK	†	†	†										
VK_FORMAT_BC7_UNORM_BLOCK	†	†	†										
VK_FORMAT_BC7_SRGB_BLOCK	†	†	†										
The VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT, VK_FORMAT_FEATURE_BLIT_SRC_BIT and VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT features must be supported in optimalTilingFeatures for all the formats in at least one of: this table, Mandatory Format Support: ETC2 and EAC Compressed Formats With VkImageType VK_IMAGE_TYPE_2D , or Mandatory Format Support: ASTC LDR Compressed Formats With VkImageType VK_IMAGE_TYPE_2D .													

Table 70. Mandatory Format Support: ETC2 and EAC Compressed Formats With `VkImageType` `VK_IMAGE_TYPE_2D`

	VK_FORMAT_FEATURE_2_COPY_IMAGE_INDIRECT_DST_BIT_KHR													
	VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_ATOMIC_BIT													
	VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT													
	VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT													
	VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT													
	VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT													
	VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT													
	VK_FORMAT_FEATURE_BLIT_DST_BIT													
	VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT													
	VK_FORMAT_FEATURE_STORAGE_IMAGE_ATOMIC_BIT													
	VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT													
	VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT													
	VK_FORMAT_FEATURE_BLIT_SRC_BIT													
	VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT													
Format														
VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK	†	†	†											
VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK	†	†	†											
VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK	†	†	†											
VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK	†	†	†											
VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK	†	†	†											
VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK	†	†	†											
VK_FORMAT_EAC_R11_UNORM_BLOCK	†	†	†											
VK_FORMAT_EAC_R11_SNORM_BLOCK	†	†	†											
VK_FORMAT_EAC_R11G11_UNORM_BLOCK	†	†	†											
VK_FORMAT_EAC_R11G11_SNORM_BLOCK	†	†	†											

The `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`, `VK_FORMAT_FEATURE_BLIT_SRC_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` features **must** be supported in `optimalTilingFeatures` for all the formats in at least one of: this table, [Mandatory Format Support: BC Compressed Formats With `VkImageType` `VK\_IMAGE\_TYPE\_2D` and `VK\_IMAGE\_TYPE\_3D`](#), or [Mandatory Format Support: ASTC LDR Compressed Formats With `VkImageType` `VK\_IMAGE\_TYPE\_2D`](#).

Table 71. Mandatory Format Support: ASTC LDR Compressed Formats With `VkImageType VK_IMAGE_TYPE_2D`[illegible]

VK_FORMAT_ASTC_10x8_UNORM_BLOCK	†	†	†											
VK_FORMAT_ASTC_10x8_SRGB_BLOCK	†	†	†											
VK_FORMAT_ASTC_10x10_UNORM_BLOCK	†	†	†											
VK_FORMAT_ASTC_10x10_SRGB_BLOCK	†	†	†											
VK_FORMAT_ASTC_12x10_UNORM_BLOCK	†	†	†											
VK_FORMAT_ASTC_12x10_SRGB_BLOCK	†	†	†											
VK_FORMAT_ASTC_12x12_UNORM_BLOCK	†	†	†											
VK_FORMAT_ASTC_12x12_SRGB_BLOCK	†	†	†											

The `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`, `VK_FORMAT_FEATURE_BLIT_SRC_BIT` and `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` features **must** be supported in `optimalTilingFeatures` for all the formats in at least one of: this table, [Mandatory Format Support: BC Compressed Formats With VkImageType VK_IMAGE_TYPE_2D](#) and `VK_IMAGE_TYPE_3D`, or [Mandatory Format Support: ETC2 and EAC Compressed Formats With VkImageType VK_IMAGE_TYPE_2D](#).

If cubic filtering is supported, `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT` **must** be supported for the following image view types:

- `VK_IMAGE_VIEW_TYPE_2D`
- `VK_IMAGE_VIEW_TYPE_2D_ARRAY`

for the following formats:

- `VK_FORMAT_R4G4_UNORM_PACK8`
- `VK_FORMAT_R4G4B4A4_UNORM_PACK16`
- `VK_FORMAT_B4G4R4A4_UNORM_PACK16`
- `VK_FORMAT_R5G6B5_UNORM_PACK16`
- `VK_FORMAT_B5G6R5_UNORM_PACK16`
- `VK_FORMAT_R5G5B5A1_UNORM_PACK16`
- `VK_FORMAT_B5G5R5A1_UNORM_PACK16`
- `VK_FORMAT_A1R5G5B5_UNORM_PACK16`
- `VK_FORMAT_R8_UNORM`
- `VK_FORMAT_R8_SNORM`
- `VK_FORMAT_R8_SRGB`
- `VK_FORMAT_R8G8_UNORM`
- `VK_FORMAT_R8G8_SNORM`
- `VK_FORMAT_R8G8_SRGB`
- `VK_FORMAT_R8G8B8_UNORM`
- `VK_FORMAT_R8G8B8_SNORM`
- `VK_FORMAT_R8G8B8_SRGB`

- `VK_FORMAT_B8G8R8_UNORM`
- `VK_FORMAT_B8G8R8_SNORM`
- `VK_FORMAT_B8G8R8_SRGB`
- `VK_FORMAT_R8G8B8A8_UNORM`
- `VK_FORMAT_R8G8B8A8_SNORM`
- `VK_FORMAT_R8G8B8A8_SRGB`
- `VK_FORMAT_B8G8R8A8_UNORM`
- `VK_FORMAT_B8G8R8A8_SNORM`
- `VK_FORMAT_B8G8R8A8_SRGB`
- `VK_FORMAT_A8B8G8R8_UNORM_PACK32`
- `VK_FORMAT_A8B8G8R8_SNORM_PACK32`
- `VK_FORMAT_A8B8G8R8_SRGB_PACK32`

If ETC compressed formats are supported, `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT` **must** be supported for the following image view types:

- `VK_IMAGE_VIEW_TYPE_2D`
- `VK_IMAGE_VIEW_TYPE_2D_ARRAY`

for the following additional formats:

- `VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK`
- `VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK`
- `VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK`
- `VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK`
- `VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK`
- `VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK`

If cubic filtering is supported for any other formats, the following image view types **must** be supported for those formats:

- `VK_IMAGE_VIEW_TYPE_2D`
- `VK_IMAGE_VIEW_TYPE_2D_ARRAY`

To be used with `VkImageView` with `subresourceRange.aspectMask` equal to `VK_IMAGE_ASPECT_COLOR_BIT`, `sampler Y'CBCR conversion` **must** be enabled for the following formats:

VK_FORMAT_G16B16G16R16_422_UNORM	1																		
VK_FORMAT_B16G16R16G16_422_UNORM	1																		
VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM	3																		
VK_FORMAT_G16_B16R16_2PLANE_420_UNORM	2																		
VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM	3																		
VK_FORMAT_G16_B16R16_2PLANE_422_UNORM	2																		
VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM	3																		
VK_FORMAT_G8_B8R8_2PLANE_444_UNORM	2																		
VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16	2																		
VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16	2																		
VK_FORMAT_G16_B16R16_2PLANE_444_UNORM	2																		
Format features marked † must be supported for optimalTilingFeatures with VkImageType VK_IMAGE_TYPE_2D if the VkPhysicalDevice supports the VkPhysicalDeviceSamplerYcbcrConversionFeatures feature.																			

Implementations are not required to support the [VK_IMAGE_CREATE_SPARSE_BINDING_BIT](#), [VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT](#), or [VK_IMAGE_CREATE_SPARSE_ALIASED_BIT](#) [VkImageCreateFlags](#) for the above formats that require [sampler Y'C_BC_R conversion](#). To determine whether the implementation supports sparse image creation flags with these formats use [vkGetPhysicalDeviceImageFormatProperties](#) or [vkGetPhysicalDeviceImageFormatProperties2](#).

[VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR](#) **must** be supported for the following formats if the [attachmentFragmentShadingRate](#) feature is supported:

- [VK_FORMAT_R8_UINT](#)

35.3.1. Formats Without Shader Storage Format

The device-level features for using a storage image or a storage texel buffer with an image format of [Unknown](#), [shaderStorageImageReadWithoutFormat](#) and [shaderStorageImageWriteWithoutFormat](#), only apply to the following formats:

- [VK_FORMAT_R8G8B8A8_UNORM](#)
- [VK_FORMAT_R8G8B8A8_SNORM](#)
- [VK_FORMAT_R8G8B8A8_UINT](#)
- [VK_FORMAT_R8G8B8A8_SINT](#)
- [VK_FORMAT_R32_UINT](#)
- [VK_FORMAT_R32_SINT](#)
- [VK_FORMAT_R32_SFLOAT](#)
- [VK_FORMAT_R32G32_UINT](#)
- [VK_FORMAT_R32G32_SINT](#)

- VK_FORMAT_R32G32_SFLOAT
- VK_FORMAT_R32G32B32A32_UINT
- VK_FORMAT_R32G32B32A32_SINT
- VK_FORMAT_R32G32B32A32_SFLOAT
- VK_FORMAT_R16G16B16A16_UINT
- VK_FORMAT_R16G16B16A16_SINT
- VK_FORMAT_R16G16B16A16_SFLOAT
- VK_FORMAT_R16G16_SFLOAT
- VK_FORMAT_B10G11R11_UFLOAT_PACK32
- VK_FORMAT_R16_SFLOAT
- VK_FORMAT_R16G16B16A16_UNORM
- VK_FORMAT_A2B10G10R10_UNORM_PACK32
- VK_FORMAT_R16G16_UNORM
- VK_FORMAT_R8G8_UNORM
- VK_FORMAT_R16_UNORM
- VK_FORMAT_R8_UNORM
- VK_FORMAT_R16G16B16A16_SNORM
- VK_FORMAT_R16G16_SNORM
- VK_FORMAT_R8G8_SNORM
- VK_FORMAT_R16_SNORM
- VK_FORMAT_R8_SNORM
- VK_FORMAT_R16G16_SINT
- VK_FORMAT_R8G8_SINT
- VK_FORMAT_R16_SINT
- VK_FORMAT_R8_SINT
- VK_FORMAT_A2B10G10R10_UINT_PACK32
- VK_FORMAT_R16G16_UINT
- VK_FORMAT_R8G8_UINT
- VK_FORMAT_R16_UINT
- VK_FORMAT_R8_UINT

NOTE

This list of formats is the union of required storage formats from [Required Format Support](#) section and formats listed in [shaderStorageImageExtendedFormats](#).

35.3.2. Format Feature Dependent Usage Flags

Certain resource usage flags depend on support for the corresponding format feature flag for the format in question. The following tables list the [VkBufferUsageFlagBits](#) and [VkImageUsageFlagBits](#) that have such dependencies, and the format feature flags they depend on. Additional restrictions, including, but not limited to, further required format feature flags specific to the particular use of the resource **may** apply, as described in the respective sections of this specification.

Table 73. Format Feature Dependent Buffer Usage Flags

Buffer usage flag	Required format feature flag
VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT	VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT
VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT	VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT
VK_BUFFER_USAGE_VERTEX_BUFFER_BIT	VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT

Table 74. Format Feature Dependent Image Usage Flags

Image usage flag	Required format feature flag
VK_IMAGE_USAGE_SAMPLED_BIT	VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT
VK_IMAGE_USAGE_STORAGE_BIT	VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT
VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT	VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT
VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT	VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT	VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT or VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT
VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR	VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR

Chapter 36. Additional Capabilities

This chapter describes additional capabilities beyond the minimum capabilities described in the [Limits](#) and [Formats](#) chapters, including:

- [Additional Image Capabilities](#)
- [Additional Buffer Capabilities](#)
- [Optional Semaphore Capabilities](#)
- [Optional Fence Capabilities](#)
- [Timestamp Calibration Capabilities](#)

36.1. Additional Image Capabilities

Additional image capabilities, such as larger dimensions or additional sample counts for certain image types, or additional capabilities for *linear* tiling format images, are described in this section.

To query additional capabilities specific to image types, call:

WARNING

This functionality is superseded by [Vulkan Version 1.1](#). See [Legacy Functionality](#) for more information.

```
// Provided by VK_VERSION_1_0
VkResult vkGetPhysicalDeviceImageFormatProperties(
    VkPhysicalDevice    physicalDevice,
    VkFormat             format,
    VkImageType         type,
    VkImageTiling        tiling,
    VkImageUsageFlags    usage,
    VkImageCreateFlags    flags,
    VkImageFormatProperties* pImageFormatProperties);
```

- `physicalDevice` is the physical device from which to query the image capabilities.
- `format` is a [VkFormat](#) value specifying the image format, corresponding to [VkImageCreateInfo::format](#).
- `type` is a [VkImageType](#) value specifying the image type, corresponding to [VkImageCreateInfo::imageType](#).
- `tiling` is a [VkImageTiling](#) value specifying the image tiling, corresponding to [VkImageCreateInfo::tiling](#).
- `usage` is a bitmask of [VkImageUsageFlagBits](#) specifying the intended usage of the image, corresponding to [VkImageCreateInfo::usage](#).
- `flags` is a bitmask of [VkImageCreateFlagBits](#) specifying additional parameters of the image, corresponding to [VkImageCreateInfo::flags](#).
- `pImageFormatProperties` is a pointer to a [VkImageFormatProperties](#) structure in which

capabilities are returned.

The `format`, `type`, `tiling`, `usage`, and `flags` parameters correspond to parameters that would be consumed by `vkCreateImage` (as members of `VkImageCreateInfo`).

If `format` is not a supported image format, or if the combination of `format`, `type`, `tiling`, `usage`, and `flags` is not supported for images, then `vkGetPhysicalDeviceImageFormatProperties` returns `VK_ERROR_FORMAT_NOT_SUPPORTED`.

The limitations on an image format that are reported by `vkGetPhysicalDeviceImageFormatProperties` have the following property: if `usage1` and `usage2` of type `VkImageUsageFlags` are such that the bits set in `usage1` are a subset of the bits set in `usage2`, and `flags1` and `flags2` of type `VkImageCreateFlags` are such that the bits set in `flags1` are a subset of the bits set in `flags2`, then the limitations for `usage1` and `flags1` **must** be no more strict than the limitations for `usage2` and `flags2`, for all values of `format`, `type`, and `tiling`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDeviceImageFormatProperties` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetPhysicalDeviceImageFormatProperties-tiling-02248
`tiling` **must** not be `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`. (Use `vkGetPhysicalDeviceImageFormatProperties2` instead)

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceImageFormatProperties-physicalDevice-parameter
`physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceImageFormatProperties-format-parameter
`format` **must** be a valid `VkFormat` value
- VUID-vkGetPhysicalDeviceImageFormatProperties-type-parameter
`type` **must** be a valid `VkImageType` value
- VUID-vkGetPhysicalDeviceImageFormatProperties-tiling-parameter
`tiling` **must** be a valid `VkImageTiling` value
- VUID-vkGetPhysicalDeviceImageFormatProperties-usage-parameter
`usage` **must** be a valid combination of `VkImageUsageFlagBits` values
- VUID-vkGetPhysicalDeviceImageFormatProperties-usage-requiredbitmask
`usage` **must** not be `0`
- VUID-vkGetPhysicalDeviceImageFormatProperties-flags-parameter
`flags` **must** be a valid combination of `VkImageCreateFlagBits` values
- VUID-vkGetPhysicalDeviceImageFormatProperties-pImageFormatProperties-parameter
`pImageFormatProperties` **must** be a valid pointer to a `VkImageFormatProperties` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_FORMAT_NOT_SUPPORTED`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkImageFormatProperties` structure is defined as:

```
// Provided by VK_VERSION_1_0
typedef struct VkImageFormatProperties {
    VkExtent3D          maxExtent;
    uint32_t            maxMipLevels;
    uint32_t            maxArrayLayers;
    VkSampleCountFlags  sampleCounts;
    VkDeviceSize        maxResourceSize;
} VkImageFormatProperties;
```

- `maxExtent` are the maximum image dimensions. See the [Allowed Extent Values](#) section below for how these values are constrained by `type`.
- `maxMipLevels` is the maximum number of mipmap levels. `maxMipLevels` **must** be equal to the number of levels in the complete mipmap chain based on the `maxExtent.width`, `maxExtent.height`, and `maxExtent.depth`, except when one of the following conditions is true, in which case it **may** instead be 1:
 - `vkGetPhysicalDeviceImageFormatProperties::tiling` was `VK_IMAGE_TILING_LINEAR`
 - `VkPhysicalDeviceImageFormatInfo2::tiling` was `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`
 - the `VkPhysicalDeviceImageFormatInfo2::pNext` chain included a `VkPhysicalDeviceExternalImageFormatInfo` structure with a handle type included in the `handleTypes` member for which mipmap image support is not required
 - image `format` is one of the [formats that require a sampler Y'C_BC_R conversion](#)
- `maxArrayLayers` is the maximum number of array layers. `maxArrayLayers` **must** be no less than `VkPhysicalDeviceLimits::maxImageArrayLayers`, except when one of the following conditions is true, in which case it **may** instead be 1:
 - `tiling` is `VK_IMAGE_TILING_LINEAR`
 - `tiling` is `VK_IMAGE_TILING_OPTIMAL` and `type` is `VK_IMAGE_TYPE_3D`
 - `format` is one of the [formats that require a sampler Y'C_BC_R conversion](#)

- If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then `maxArrayLayers` **must** not be 0.
- `sampleCounts` is a bitmask of `VkSampleCountFlagBits` specifying all the supported sample counts for this image as described [below](#).
- `maxResourceSize` is an upper bound on the total image size in bytes, inclusive of all image subresources. Implementations **may** have an address space limit on total size of a resource, which is advertised by this property. `maxResourceSize` **must** be at least 2^{31} .

NOTE

There is no mechanism to query the size of an image before creating it, to compare that size against `maxResourceSize`. If an application attempts to create an image that exceeds this limit, the creation will fail and `vkCreateImage` will return `VK_ERROR_OUT_OF_DEVICE_MEMORY`. While the advertised limit **must** be at least 2^{31} , it **may** not be possible to create an image that approaches that size, particularly for `VK_IMAGE_TYPE_1D`.

If the combination of parameters to `vkGetPhysicalDeviceImageFormatProperties` is not supported by the implementation for use in `vkCreateImage`, then all members of `VkImageFormatProperties` will be filled with zero.

NOTE

Filling `VkImageFormatProperties` with zero for unsupported formats is an exception to the usual rule that output structures have undefined contents on error. This exception was unintentional, but is preserved for backwards compatibility.

To query additional capabilities specific to image types, call:

```
// Provided by VK_VERSION_1_1
VkResult vkGetPhysicalDeviceImageFormatProperties2(
    VkPhysicalDevice          physicalDevice,
    const VkPhysicalDeviceImageFormatInfo2* pImageFormatInfo,
    VkImageFormatProperties2* pImageFormatProperties);
```

- `physicalDevice` is the physical device from which to query the image capabilities.
- `pImageFormatInfo` is a pointer to a `VkPhysicalDeviceImageFormatInfo2` structure describing the parameters that would be consumed by `vkCreateImage`.
- `pImageFormatProperties` is a pointer to a `VkImageFormatProperties2` structure in which capabilities are returned.

`vkGetPhysicalDeviceImageFormatProperties2` behaves similarly to `vkGetPhysicalDeviceImageFormatProperties`, with the ability to return extended information in a `pNext` chain of output structures.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDeviceImageFormatProperties2` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceImageFormatProperties2-physicalDevice-parameter

`physicalDevice` **must** be a valid `VkPhysicalDevice` handle

- VUID-vkGetPhysicalDeviceImageFormatProperties2-pImageFormatInfo-parameter `pImageFormatInfo` **must** be a valid pointer to a valid `VkPhysicalDeviceImageFormatInfo2` structure
- VUID-vkGetPhysicalDeviceImageFormatProperties2-pImageFormatProperties-parameter `pImageFormatProperties` **must** be a valid pointer to a `VkImageFormatProperties2` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_FORMAT_NOT_SUPPORTED`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkPhysicalDeviceImageFormatInfo2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceImageFormatInfo2 {
    VkStructureType    sType;
    const void*        pNext;
    VkFormat            format;
    VkImageType         type;
    VkImageTiling        tiling;
    VkImageUsageFlags   usage;
    VkImageCreateFlags   flags;
} VkPhysicalDeviceImageFormatInfo2;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure. The `pNext` chain of `VkPhysicalDeviceImageFormatInfo2` is used to provide additional image parameters to `vkGetPhysicalDeviceImageFormatProperties2`.
- `format` is a `VkFormat` value indicating the image format, corresponding to `VkImageCreateInfo::format`.
- `type` is a `VkImageType` value indicating the image type, corresponding to `VkImageCreateInfo::imageType`.
- `tiling` is a `VkImageTiling` value indicating the image tiling, corresponding to `VkImageCreateInfo::tiling`.

- **usage** is a bitmask of `VkImageUsageFlagBits` indicating the intended usage of the image, corresponding to `VkImageCreateInfo::usage`.
- **flags** is a bitmask of `VkImageCreateFlagBits` indicating additional parameters of the image, corresponding to `VkImageCreateInfo::flags`.

The members of `VkPhysicalDeviceImageFormatInfo2` correspond to the arguments to `vkGetPhysicalDeviceImageFormatProperties`, with **sType** and **pNext** added for extensibility.

Valid Usage

- VUID-VkPhysicalDeviceImageFormatInfo2-tiling-02249
tiling **must** be `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` if and only if the **pNext** chain includes `VkPhysicalDeviceImageDrmFormatModifierInfoEXT`
- VUID-VkPhysicalDeviceImageFormatInfo2-tiling-02313
If **tiling** is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` and **flags** contains `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`, then the **pNext** chain **must** include a `VkImageFormatListCreateInfo` structure with non-zero **viewFormatCount**

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageFormatInfo2-sType-sType
sType **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_FORMAT_INFO_2`
- VUID-VkPhysicalDeviceImageFormatInfo2-pNext-pNext
Each **pNext** member of any structure (including this one) in the **pNext** chain **must** be either `NULL` or a pointer to a valid instance of `VkImageFormatListCreateInfo`, `VkImageStencilUsageCreateInfo`, `VkPhysicalDeviceExternalImageFormatInfo`, `VkPhysicalDeviceImageDrmFormatModifierInfoEXT`, or `VkPhysicalDeviceImageViewImageFormatInfoEXT`
- VUID-VkPhysicalDeviceImageFormatInfo2-sType-unique
The **sType** value of each structure in the **pNext** chain **must** be unique
- VUID-VkPhysicalDeviceImageFormatInfo2-format-parameter
format **must** be a valid `VkFormat` value
- VUID-VkPhysicalDeviceImageFormatInfo2-type-parameter
type **must** be a valid `VkImageType` value
- VUID-VkPhysicalDeviceImageFormatInfo2-tiling-parameter
tiling **must** be a valid `VkImageTiling` value
- VUID-VkPhysicalDeviceImageFormatInfo2-usage-parameter
usage **must** be a valid combination of `VkImageUsageFlagBits` values
- VUID-VkPhysicalDeviceImageFormatInfo2-usage-requiredbitmask
usage **must** not be `0`
- VUID-VkPhysicalDeviceImageFormatInfo2-flags-parameter
flags **must** be a valid combination of `VkImageCreateFlagBits` values

The `VkImageFormatProperties2` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkImageFormatProperties2 {
    VkStructureType    sType;
    void*              pNext;
    VkImageFormatProperties    imageFormatProperties;
} VkImageFormatProperties2;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure. The `pNext` chain of `VkImageFormatProperties2` is used to allow the specification of additional capabilities to be returned from `vkGetPhysicalDeviceImageFormatProperties2`.
- `imageFormatProperties` is a `VkImageFormatProperties` structure in which capabilities are returned.

If the combination of parameters to `vkGetPhysicalDeviceImageFormatProperties2` is not supported by the implementation for use in `vkCreateImage`, then all members of `imageFormatProperties` will be filled with zero.

NOTE

Filling `imageFormatProperties` with zero for unsupported formats is an exception to the usual rule that output structures have undefined contents on error. This exception was unintentional, but is preserved for backwards compatibility. This exception only applies to `imageFormatProperties`, not `sType`, `pNext`, or any structures chained from `pNext`.

Valid Usage (Implicit)

- VUID-VkImageFormatProperties2-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_FORMAT_PROPERTIES_2`
- VUID-VkImageFormatProperties2-pNext-pNext
Each `pNext` member of any structure (including this one) in the `pNext` chain **must** be either `NULL` or a pointer to a valid instance of `VkExternalImageFormatProperties`, `VkFilterCubicImageViewImageFormatPropertiesEXT`, or `VkSamplerYcbcrConversionImageFormatProperties`
- VUID-VkImageFormatProperties2-sType-unique
The `sType` value of each structure in the `pNext` chain **must** be unique

To determine the image capabilities compatible with an external memory handle type, add a `VkPhysicalDeviceExternalImageFormatInfo` structure to the `pNext` chain of the `VkPhysicalDeviceImageFormatInfo2` structure and a `VkExternalImageFormatProperties` structure to the `pNext` chain of the `VkImageFormatProperties2` structure.

The `VkPhysicalDeviceExternalImageFormatInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceExternalImageFormatInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkExternalMemoryHandleTypeFlagBits handleType;
} VkPhysicalDeviceExternalImageFormatInfo;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `handleType` is a `VkExternalMemoryHandleTypeFlagBits` value specifying the memory handle type that will be used with the memory associated with the image.

If `handleType` is 0, `vkGetPhysicalDeviceImageFormatProperties2` will behave as if `VkPhysicalDeviceExternalImageFormatInfo` was not present, and `VkExternalImageFormatProperties` will be ignored.

If `handleType` is not compatible with the `format`, `type`, `tiling`, `usage`, and `flags` specified in `VkPhysicalDeviceImageFormatInfo2`, then `vkGetPhysicalDeviceImageFormatProperties2` returns `VK_ERROR_FORMAT_NOT_SUPPORTED`.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalImageFormatInfo-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_IMAGE_FORMAT_INFO`
- VUID-VkPhysicalDeviceExternalImageFormatInfo-handleType-parameter
If `handleType` is not 0, `handleType` **must** be a valid `VkExternalMemoryHandleTypeFlagBits` value

Possible values of `VkPhysicalDeviceExternalImageFormatInfo::handleType`, specifying an external memory handle type, are:

```

// Provided by VK_VERSION_1_1
typedef enum VkExternalMemoryHandleTypeFlagBits {
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT = 0x00000001,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT = 0x00000002,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT = 0x00000004,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT = 0x00000008,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT = 0x00000010,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT = 0x00000020,
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT = 0x00000040,
// Provided by VK_EXT_external_memory_dma_buf
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT = 0x00000200,
// Provided by VK_EXT_external_memory_host
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT = 0x00000080,
// Provided by VK_EXT_external_memory_host
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT = 0x00000100,
// Provided by VK_NV_external_memory_sci_buf
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV = 0x00002000,
// Provided by VK_QNX_external_memory_screen_buffer
    VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX = 0x00004000,
} VkExternalMemoryHandleTypeFlagBits;

```

- **VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT** specifies a POSIX file descriptor handle that has only limited valid usage outside of Vulkan and other compatible APIs. It **must** be compatible with the POSIX system calls **dup**, **dup2**, **close**, and the non-standard system call **dup3**. Additionally, it **must** be transportable over a socket using an **SCM_RIGHTS** control message. It owns a reference to the underlying memory resource represented by its Vulkan memory object.
- **VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT** specifies an NT handle that has only limited valid usage outside of Vulkan and other compatible APIs. It **must** be compatible with the functions **DuplicateHandle**, **CloseHandle**, **CompareObjectHandles**, **GetHandleInformation**, and **SetHandleInformation**. It owns a reference to the underlying memory resource represented by its Vulkan memory object.
- **VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT** specifies a global share handle that has only limited valid usage outside of Vulkan and other compatible APIs. It is not compatible with any native APIs. It does not own a reference to the underlying memory resource represented by its Vulkan memory object, and will therefore become invalid when all Vulkan memory objects associated with it are destroyed.
- **VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT** specifies an NT handle returned by **IDXGIResource1::CreateSharedHandle** referring to a Direct3D 10 or 11 texture resource. It owns a reference to the memory used by the Direct3D resource.
- **VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT** specifies a global share handle returned by **IDXGIResource::GetSharedHandle** referring to a Direct3D 10 or 11 texture resource. It does not own a reference to the underlying Direct3D resource, and will therefore become invalid when all Vulkan memory objects and Direct3D resources associated with it are destroyed.
- **VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT** specifies an NT handle returned by **ID3D12Device::CreateSharedHandle** referring to a Direct3D 12 heap resource. It owns a reference to the resources used by the Direct3D heap.

- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT` specifies an NT handle returned by `ID3D12Device::CreateSharedHandle` referring to a Direct3D 12 committed resource. It owns a reference to the memory used by the Direct3D resource.
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT` specifies a host pointer returned by a host memory allocation command. It does not own a reference to the underlying memory resource, and will therefore become invalid if the host memory is freed.
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT` specifies a host pointer to *host mapped foreign memory*. It does not own a reference to the underlying memory resource, and will therefore become invalid if the foreign memory is unmapped or otherwise becomes no longer available.
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT` is a file descriptor for a Linux `dma_buf`. It owns a reference to the underlying memory resource represented by its Vulkan memory object.
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV` specifies a volatile memory object (`NvSciBufObj`) that is backed by a buffer and shareable across various hardware engines including the CPU, and software (intra-process and inter-process) and hardware (system memory) operating domains.
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX` specifies a `_screen_buffer` object defined by the QNX SDP. See [QNX Screen Buffer](#) for more details of this handle type.

Some external memory handle types can only be shared within the same underlying physical device and/or the same driver version, as defined in the following table:

Table 75. External Memory Handle Types Compatibility

Handle type	VkPhysicalDeviceIDProperties::driverUUID	VkPhysicalDeviceIDProperties::deviceUUID
VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_FD_BIT	Must match	Must match
VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_BIT	Must match	Must match
VK_EXTERNAL_MEMORY_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT	Must match	Must match
VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT	Must match	Must match
VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT	Must match	Must match
VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_HEAP_BIT	Must match	Must match
VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT	Must match	Must match
VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT	No restriction	No restriction
VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT	No restriction	No restriction
VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT	No restriction	No restriction
VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV	No restriction	No restriction
VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX	No restriction	No restriction

NOTE

The above table does not restrict the drivers and devices with which [VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT](#) and [VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT](#) **may** be shared, as these handle types inherently mean memory that does not come from the same device, as they import memory from the host or a foreign device, respectively.

NOTE

Even though the above table does not restrict the drivers and devices with which [VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT](#) **may** be shared, query mechanisms exist in the Vulkan API that prevent the import of incompatible dma-bufs (such as [vkGetMemoryFdPropertiesKHR](#)) and that prevent incompatible usage of dma-bufs (such as [VkPhysicalDeviceExternalBufferInfo](#) and [VkPhysicalDeviceExternalImageFormatInfo](#)).

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkExternalMemoryHandleTypeFlags;
```

`VkExternalMemoryHandleTypeFlags` is a bitmask type for setting a mask of zero or more `VkExternalMemoryHandleTypeFlagBits`.

The `VkExternalImageFormatProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalImageFormatProperties {
    VkStructureType          sType;
    void*                    pNext;
    VkExternalMemoryProperties externalMemoryProperties;
} VkExternalImageFormatProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `externalMemoryProperties` is a `VkExternalMemoryProperties` structure specifying various capabilities of the external handle type when used with the specified image creation parameters.

Valid Usage (Implicit)

- VUID-VkExternalImageFormatProperties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_IMAGE_FORMAT_PROPERTIES`

The `VkExternalMemoryProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalMemoryProperties {
    VkExternalMemoryFeatureFlags    externalMemoryFeatures;
    VkExternalMemoryHandleTypeFlags exportFromImportedHandleTypes;
    VkExternalMemoryHandleTypeFlags compatibleHandleTypes;
} VkExternalMemoryProperties;
```

- `externalMemoryFeatures` is a bitmask of `VkExternalMemoryFeatureFlagBits` specifying the features of `handleType`.
- `exportFromImportedHandleTypes` is a bitmask of `VkExternalMemoryHandleTypeFlagBits` specifying which types of imported handle `handleType` **can** be exported from.
- `compatibleHandleTypes` is a bitmask of `VkExternalMemoryHandleTypeFlagBits` specifying handle types which **can** be specified at the same time as `handleType` when creating an image compatible with external memory.

`compatibleHandleTypes` **must** include at least `handleType`. Inclusion of a handle type in

`compatibleHandleTypes` does not imply the values returned in `VkImageFormatProperties2` will be the same when `VkPhysicalDeviceExternalImageFormatInfo::handleType` is set to that type. The application is responsible for querying the capabilities of all handle types intended for concurrent use in a single image and intersecting them to obtain the compatible set of capabilities.

Bits which **may** be set in `VkExternalMemoryProperties::externalMemoryFeatures`, specifying features of an external memory handle type, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkExternalMemoryFeatureFlagBits {
    VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT = 0x00000001,
    VK_EXTERNAL_MEMORY_FEATURE_EXPORTABLE_BIT = 0x00000002,
    VK_EXTERNAL_MEMORY_FEATURE_IMPORTABLE_BIT = 0x00000004,
} VkExternalMemoryFeatureFlagBits;
```

- `VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT` specifies that images or buffers created with the specified parameters and handle type **must** use the mechanisms defined by `VkMemoryDedicatedRequirements` and `VkMemoryDedicatedAllocateInfo` to create (or import) a dedicated allocation for the image or buffer.
- `VK_EXTERNAL_MEMORY_FEATURE_EXPORTABLE_BIT` specifies that handles of this type **can** be exported from Vulkan memory objects.
- `VK_EXTERNAL_MEMORY_FEATURE_IMPORTABLE_BIT` specifies that handles of this type **can** be imported as Vulkan memory objects.

Because their semantics in external APIs roughly align with that of an image or buffer with a dedicated allocation in Vulkan, implementations are **required** to report `VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT` for the following external handle types:

- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_BIT`
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D11_TEXTURE_KMT_BIT`
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_D3D12_RESOURCE_BIT`
- `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX` for images only

Implementations **must** not report `VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT` for buffers with external handle type `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX`. Implementations **must** not report `VK_EXTERNAL_MEMORY_FEATURE_DEDICATED_ONLY_BIT` for images or buffers with external handle type `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT`, or `VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT`.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkExternalMemoryFeatureFlags;
```

`VkExternalMemoryFeatureFlags` is a bitmask type for setting a mask of zero or more `VkExternalMemoryFeatureFlagBits`.

To query the image capabilities that are compatible with a `Linux DRM format modifier`, set

VkPhysicalDeviceImageFormatInfo2::tiling to VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT and add a VkPhysicalDeviceImageDrmFormatModifierInfoEXT structure to the pNext chain of VkPhysicalDeviceImageFormatInfo2.

The VkPhysicalDeviceImageDrmFormatModifierInfoEXT structure is defined as:

```
// Provided by VK_EXT_image_drm_format_modifier
typedef struct VkPhysicalDeviceImageDrmFormatModifierInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    uint64_t           drmFormatModifier;
    VkSharingMode       sharingMode;
    uint32_t           queueFamilyIndexCount;
    const uint32_t*     pQueueFamilyIndices;
} VkPhysicalDeviceImageDrmFormatModifierInfoEXT;
```

- sType is a VkStructureType value identifying this structure.
- pNext is NULL or a pointer to a structure extending this structure.
- drmFormatModifier is the image's Linux DRM format modifier, corresponding to VkImageDrmFormatModifierExplicitCreateInfoEXT::drmFormatModifier or to VkImageDrmFormatModifierListCreateInfoEXT::pDrmFormatModifiers.
- sharingMode specifies how the image will be accessed by multiple queue families.
- queueFamilyIndexCount is the number of entries in the pQueueFamilyIndices array.
- pQueueFamilyIndices is a pointer to an array of queue families that will access the image. It is ignored if sharingMode is not VK_SHARING_MODE_CONCURRENT.

If the drmFormatModifier is incompatible with the parameters specified in VkPhysicalDeviceImageFormatInfo2 and its pNext chain, then vkGetPhysicalDeviceImageFormatProperties2 returns VK_ERROR_FORMAT_NOT_SUPPORTED. The implementation **must** support the query of any drmFormatModifier, including unknown and invalid modifier values.

Valid Usage

- VUID-VkPhysicalDeviceImageDrmFormatModifierInfoEXT-sharingMode-02314
If sharingMode is VK_SHARING_MODE_CONCURRENT, then pQueueFamilyIndices **must** be a valid pointer to an array of queueFamilyIndexCount uint32_t values
- VUID-VkPhysicalDeviceImageDrmFormatModifierInfoEXT-sharingMode-02315
If sharingMode is VK_SHARING_MODE_CONCURRENT, then queueFamilyIndexCount **must** be greater than 1
- VUID-VkPhysicalDeviceImageDrmFormatModifierInfoEXT-sharingMode-02316
If sharingMode is VK_SHARING_MODE_CONCURRENT, each element of pQueueFamilyIndices **must** be unique and **must** be less than the pQueueFamilyPropertyCount returned by vkGetPhysicalDeviceQueueFamilyProperties2 for the physicalDevice that was used to create device

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageDrmFormatModifierInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_DRM_FORMAT_MODIFIER_INFO_EXT`
- VUID-VkPhysicalDeviceImageDrmFormatModifierInfoEXT-sharingMode-parameter
`sharingMode` **must** be a valid `VkSharingMode` value

To determine the number of combined image samplers required to support a `multi-planar` format, add `VkSamplerYcbcrConversionImageFormatProperties` to the `pNext` chain of the `VkImageFormatProperties2` structure in a call to `vkGetPhysicalDeviceImageFormatProperties2`.

The `VkSamplerYcbcrConversionImageFormatProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkSamplerYcbcrConversionImageFormatProperties {
    VkStructureType    sType;
    void*              pNext;
    uint32_t           combinedImageSamplerDescriptorCount;
} VkSamplerYcbcrConversionImageFormatProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `combinedImageSamplerDescriptorCount` is the number of combined image sampler descriptors that the implementation uses to access the format.

Valid Usage (Implicit)

- VUID-VkSamplerYcbcrConversionImageFormatProperties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_IMAGE_FORMAT_PROPERTIES`

`combinedImageSamplerDescriptorCount` is a number between 1 and the number of planes in the format. A descriptor set layout binding with immutable Y'C_BC_R conversion samplers will have a maximum `combinedImageSamplerDescriptorCount` which is the maximum across all formats supported by its samplers of the `combinedImageSamplerDescriptorCount` for each format. Descriptor sets with that layout will internally use that maximum `combinedImageSamplerDescriptorCount` descriptors for each descriptor in the binding. This expanded number of descriptors will be consumed from the descriptor pool when a descriptor set is allocated, and counts towards the `maxDescriptorSetSamplers`, `maxDescriptorSetSampledImages`, `maxPerStageDescriptorSamplers`, and `maxPerStageDescriptorSampledImages` limits.

NOTE

All descriptors in a binding use the same maximum `combinedImageSamplerDescriptorCount` descriptors to allow implementations to use a uniform stride for dynamic indexing of the descriptors in the binding.

For example, consider a descriptor set layout binding with two descriptors and immutable samplers for [multi-planar formats](#) that have [VkSamplerYcbcrConversionImageFormatProperties::combinedImageSamplerDescriptorCount](#) values of 2 and 3 respectively. There are two descriptors in the binding and the maximum [combinedImageSamplerDescriptorCount](#) is 3, so descriptor sets with this layout consume 6 descriptors from the descriptor pool. To create a descriptor pool that allows allocating four descriptor sets with this layout, [descriptorCount](#) must be at least 24.

To determine if cubic filtering can be used with a given image format and a given image view type add a [VkPhysicalDeviceImageViewImageFormatInfoEXT](#) structure to the [pNext](#) chain of the [VkPhysicalDeviceImageFormatInfo2](#) structure, and a [VkFilterCubicImageViewImageFormatPropertiesEXT](#) structure to the [pNext](#) chain of the [VkImageFormatProperties2](#) structure.

The [VkPhysicalDeviceImageViewImageFormatInfoEXT](#) structure is defined as:

```
// Provided by VK_EXT_filter_cubic
typedef struct VkPhysicalDeviceImageViewImageFormatInfoEXT {
    VkStructureType    sType;
    void*              pNext;
    VkImageViewType    imageViewType;
} VkPhysicalDeviceImageViewImageFormatInfoEXT;
```

- [sType](#) is a [VkStructureType](#) value identifying this structure.
- [pNext](#) is [NULL](#) or a pointer to a structure extending this structure.
- [imageViewType](#) is a [VkImageViewType](#) value specifying the type of the image view.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceImageViewImageFormatInfoEXT-sType-sType [sType](#) **must** be [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_VIEW_IMAGE_FORMAT_INFO_EXT](#)
- VUID-VkPhysicalDeviceImageViewImageFormatInfoEXT-imageViewType-parameter [imageViewType](#) **must** be a valid [VkImageViewType](#) value

The [VkFilterCubicImageViewImageFormatPropertiesEXT](#) structure is defined as:

```
// Provided by VK_EXT_filter_cubic
typedef struct VkFilterCubicImageViewImageFormatPropertiesEXT {
    VkStructureType    sType;
    void*              pNext;
    VkBool32           filterCubic;
    VkBool32           filterCubicMinmax;
} VkFilterCubicImageViewImageFormatPropertiesEXT;
```


- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `filterCubic` tells if image format, image type and image view type **can** be used with cubic filtering. This field is set by the implementation. An application-specified value is ignored.
- `filterCubicMinmax` tells if image format, image type and image view type **can** be used with cubic filtering and minmax filtering. This field is set by the implementation. An application-specified value is ignored.

Valid Usage (Implicit)

- VUID-VkFilterCubicImageViewImageFormatPropertiesEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_FILTER_CUBIC_IMAGE_VIEW_IMAGE_FORMAT_PROPERTIES_EXT`

Valid Usage

- VUID-VkFilterCubicImageViewImageFormatPropertiesEXT-pNext-02627
If the `pNext` chain of the `VkImageFormatProperties2` structure includes a `VkFilterCubicImageViewImageFormatPropertiesEXT` structure, the `pNext` chain of the `VkPhysicalDeviceImageFormatInfo2` structure **must** include a `VkPhysicalDeviceImageFormatInfoEXT` structure with an `imageViewType` that is compatible with `imageType`

36.1.1. Supported Sample Counts

`vkGetPhysicalDeviceImageFormatProperties` returns a bitmask of `VkSampleCountFlagBits` in `sampleCounts` specifying the supported sample counts for the image parameters.

`sampleCounts` will be `VK_SAMPLE_COUNT_1_BIT` if at least one of the following conditions is true:

- `tiling` is `VK_IMAGE_TILING_LINEAR`
- `type` is not `VK_IMAGE_TYPE_2D`
- `flags` contains `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`
- Neither the `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` flag nor the `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT` flag in `VkFormatProperties::optimalTilingFeatures` returned by `vkGetPhysicalDeviceFormatProperties` is set
- `VkPhysicalDeviceExternalImageFormatInfo::handleType` is an external handle type for which multisampled image support is not required.
- `format` is one of the `formats that require a sampler Y'CBCR conversion`
- `usage` contains `VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`

Otherwise, the bits set in `sampleCounts` will be the sample counts supported for the specified values of `usage` and `format`. For each bit set in `usage`, the supported sample counts relate to the limits in `VkPhysicalDeviceLimits` as follows:

- If `usage` includes `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` and `format` is a floating- or fixed-point color format, a superset of `VkPhysicalDeviceLimits::framebufferColorSampleCounts`
- If `usage` includes `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` and `format` is an integer format, a superset of `VkPhysicalDeviceVulkan12Properties::framebufferIntegerColorSampleCounts`
- If `usage` includes `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, and `format` includes a depth component, a superset of `VkPhysicalDeviceLimits::framebufferDepthSampleCounts`
- If `usage` includes `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, and `format` includes a stencil component, a superset of `VkPhysicalDeviceLimits::framebufferStencilSampleCounts`
- If `usage` includes `VK_IMAGE_USAGE_SAMPLED_BIT`, and `format` includes a color component, a superset of `VkPhysicalDeviceLimits::sampledImageColorSampleCounts`
- If `usage` includes `VK_IMAGE_USAGE_SAMPLED_BIT`, and `format` includes a depth component, a superset of `VkPhysicalDeviceLimits::sampledImageDepthSampleCounts`
- If `usage` includes `VK_IMAGE_USAGE_SAMPLED_BIT`, and `format` is an integer format, a superset of `VkPhysicalDeviceLimits::sampledImageIntegerSampleCounts`
- If `usage` includes `VK_IMAGE_USAGE_STORAGE_BIT`, a superset of `VkPhysicalDeviceLimits::storageImageSampleCounts`

If multiple bits are set in `usage`, `sampleCounts` will be the intersection of the per-usage values described above.

If none of the bits described above are set in `usage`, then there is no corresponding limit in `VkPhysicalDeviceLimits`. In this case, `sampleCounts` **must** include at least `VK_SAMPLE_COUNT_1_BIT`.

36.1.2. Allowed Extent Values Based on Image Type

Implementations **may** support extent values larger than the [required minimum/maximum values](#) for certain types of images. `VkImageFormatProperties::maxExtent` for each type is subject to the constraints below.

NOTE

Implementations **must** support images with dimensions up to the [required minimum/maximum values](#) for all types of images. It follows that the query for additional capabilities **must** return extent values that are at least as large as the required values.

For `VK_IMAGE_TYPE_1D`:

- `maxExtent.width` $\geq$ `VkPhysicalDeviceLimits::maxImageDimension1D`
- `maxExtent.height` = 1
- `maxExtent.depth` = 1

For `VK_IMAGE_TYPE_2D` when `flags` does not contain `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`:

- `maxExtent.width` $\geq$ `VkPhysicalDeviceLimits::maxImageDimension2D`
- `maxExtent.height` $\geq$ `VkPhysicalDeviceLimits::maxImageDimension2D`
- `maxExtent.depth` = 1

For `VK_IMAGE_TYPE_2D` when `flags` contains `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`:

- `maxExtent.width` $\geq$ `VkPhysicalDeviceLimits::maxImageDimensionCube`
- `maxExtent.height` $\geq$ `VkPhysicalDeviceLimits::maxImageDimensionCube`
- `maxExtent.depth` = 1

For `VK_IMAGE_TYPE_3D`:

- `maxExtent.width` $\geq$ `VkPhysicalDeviceLimits::maxImageDimension3D`
- `maxExtent.height` $\geq$ `VkPhysicalDeviceLimits::maxImageDimension3D`
- `maxExtent.depth` $\geq$ `VkPhysicalDeviceLimits::maxImageDimension3D`

36.2. Additional Buffer Capabilities

To query the external handle types supported by buffers, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceExternalBufferProperties(
    VkPhysicalDevice          physicalDevice,
    const VkPhysicalDeviceExternalBufferInfo* pExternalBufferInfo,
    VkExternalBufferProperties* pExternalBufferProperties);
```

- `physicalDevice` is the physical device from which to query the buffer capabilities.
- `pExternalBufferInfo` is a pointer to a `VkPhysicalDeviceExternalBufferInfo` structure describing the parameters that would be consumed by `vkCreateBuffer`.
- `pExternalBufferProperties` is a pointer to a `VkExternalBufferProperties` structure in which capabilities are returned.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceExternalBufferProperties-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceExternalBufferProperties-pExternalBufferInfo-parameter `pExternalBufferInfo` **must** be a valid pointer to a valid `VkPhysicalDeviceExternalBufferInfo` structure
- VUID-vkGetPhysicalDeviceExternalBufferProperties-pExternalBufferProperties-parameter `pExternalBufferProperties` **must** be a valid pointer to a `VkExternalBufferProperties` structure

The `VkPhysicalDeviceExternalBufferInfo` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceExternalBufferInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkBufferCreateFlags       flags;
    VkBufferUsageFlags        usage;
    VkExternalMemoryHandleTypeFlagBits handleType;
} VkPhysicalDeviceExternalBufferInfo;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **flags** is a bitmask of [VkBufferCreateFlagBits](#) describing additional parameters of the buffer, corresponding to [VkBufferCreateInfo::flags](#).
- **usage** is a bitmask of [VkBufferUsageFlagBits](#) describing the intended usage of the buffer, corresponding to [VkBufferCreateInfo::usage](#).
- **handleType** is a [VkExternalMemoryHandleTypeFlagBits](#) value specifying the memory handle type that will be used with the memory associated with the buffer.

Only usage flags representable in [VkBufferUsageFlagBits](#) are returned in this structure's **usage**.

Valid Usage

- VUID-VkPhysicalDeviceExternalBufferInfo-None-09499
usage must be a valid combination of [VkBufferUsageFlagBits](#) values
- VUID-VkPhysicalDeviceExternalBufferInfo-None-09500
usage must not be 0

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalBufferInfo-sType-sType
sType must be [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_BUFFER_INFO](#)
- VUID-VkPhysicalDeviceExternalBufferInfo-pNext-pNext
pNext must be **NULL**
- VUID-VkPhysicalDeviceExternalBufferInfo-flags-parameter
flags must be a valid combination of [VkBufferCreateFlagBits](#) values
- VUID-VkPhysicalDeviceExternalBufferInfo-handleType-parameter
handleType must be a valid [VkExternalMemoryHandleTypeFlagBits](#) value

The [VkExternalBufferProperties](#) structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalBufferProperties {
    VkStructureType      sType;
    void*                pNext;
    VkExternalMemoryProperties externalMemoryProperties;
} VkExternalBufferProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `externalMemoryProperties` is a `VkExternalMemoryProperties` structure specifying various capabilities of the external handle type when used with the specified buffer creation parameters.

Valid Usage (Implicit)

- VUID-VkExternalBufferProperties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_BUFFER_PROPERTIES`
- VUID-VkExternalBufferProperties-pNext-pNext
`pNext` **must** be `NULL`

36.3. Optional Semaphore Capabilities

Semaphores **may** support import and export of their `payload` to external handles. To query the external handle types supported by semaphores, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceExternalSemaphoreProperties(
    VkPhysicalDevice          physicalDevice,
    const VkPhysicalDeviceExternalSemaphoreInfo* pExternalSemaphoreInfo,
    VkExternalSemaphoreProperties* pExternalSemaphoreProperties);
```

- `physicalDevice` is the physical device from which to query the semaphore capabilities.
- `pExternalSemaphoreInfo` is a pointer to a `VkPhysicalDeviceExternalSemaphoreInfo` structure describing the parameters that would be consumed by `vkCreateSemaphore`.
- `pExternalSemaphoreProperties` is a pointer to a `VkExternalSemaphoreProperties` structure in which capabilities are returned.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceExternalSemaphoreProperties-physicalDevice-parameter
`physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceExternalSemaphoreProperties-pExternalSemaphoreInfo-

parameter
pExternalSemaphoreInfo **must** be a valid pointer to a valid **VkPhysicalDeviceExternalSemaphoreInfo** structure

- VUID-vkGetPhysicalDeviceExternalSemaphoreProperties-pExternalSemaphoreProperties-parameter
pExternalSemaphoreProperties **must** be a valid pointer to a **VkExternalSemaphoreProperties** structure

The **VkPhysicalDeviceExternalSemaphoreInfo** structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceExternalSemaphoreInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkExternalSemaphoreHandleTypeFlagBits handleType;
} VkPhysicalDeviceExternalSemaphoreInfo;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **handleType** is a **VkExternalSemaphoreHandleTypeFlagBits** value specifying the external semaphore handle type for which capabilities will be returned.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalSemaphoreInfo-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SEMAPHORE_INFO**
- VUID-VkPhysicalDeviceExternalSemaphoreInfo-pNext-pNext
pNext **must** be **NULL** or a pointer to a valid instance of **VkSemaphoreTypeCreateInfo**
- VUID-VkPhysicalDeviceExternalSemaphoreInfo-sType-unique
The **sType** value of each structure in the **pNext** chain **must** be unique
- VUID-VkPhysicalDeviceExternalSemaphoreInfo-handleType-parameter
handleType **must** be a valid **VkExternalSemaphoreHandleTypeFlagBits** value

Bits which **may** be set in **VkPhysicalDeviceExternalSemaphoreInfo::handleType**, specifying an external semaphore handle type, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkExternalSemaphoreHandleTypeFlagBits {
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT = 0x00000001,
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT = 0x00000002,
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT = 0x00000004,
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT = 0x00000008,
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT = 0x00000010,
    // Provided by VK_NV_external_sci_sync
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV = 0x00000020,
    VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D11_FENCE_BIT =
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT,
} VkExternalSemaphoreHandleTypeFlagBits;
```

- **VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT** specifies a POSIX file descriptor handle that has only limited valid usage outside of Vulkan and other compatible APIs. It **must** be compatible with the POSIX system calls **dup**, **dup2**, **close**, and the non-standard system call **dup3**. Additionally, it **must** be transportable over a socket using an **SCM_RIGHTS** control message. It owns a reference to the underlying synchronization primitive represented by its Vulkan semaphore object.
- **VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT** specifies an NT handle that has only limited valid usage outside of Vulkan and other compatible APIs. It **must** be compatible with the functions **DuplicateHandle**, **CloseHandle**, **CompareObjectHandles**, **GetHandleInformation**, and **SetHandleInformation**. It owns a reference to the underlying synchronization primitive represented by its Vulkan semaphore object.
- **VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT** specifies a global share handle that has only limited valid usage outside of Vulkan and other compatible APIs. It is not compatible with any native APIs. It does not own a reference to the underlying synchronization primitive represented by its Vulkan semaphore object, and will therefore become invalid when all Vulkan semaphore objects associated with it are destroyed.
- **VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT** specifies an NT handle returned by **ID3D12Device::CreateSharedHandle** referring to a Direct3D 12 fence, or **ID3D11Device5::CreateFence** referring to a Direct3D 11 fence. It owns a reference to the underlying synchronization primitive associated with the Direct3D fence.
- **VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D11_FENCE_BIT** is an alias of **VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT** with the same meaning. It is provided for convenience and code clarity when interacting with D3D11 fences.
- **VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT** specifies a POSIX file descriptor handle to a Linux Sync File or Android Fence object. It can be used with any native API accepting a valid sync file or fence as input. It owns a reference to the underlying synchronization primitive associated with the file descriptor. Implementations which support importing this handle type **must** accept any type of sync or fence FD supported by the native system they are running on.
- **VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV** specifies a synchronization object (**NvSciSyncObj**) shareable across various hardware engines including the CPU and software (intra-process and inter-process) operating domains and perform signal and wait operations.

NOTE | Handles of type **VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT** generated by the

implementation may represent either Linux Sync Files or Android Fences at the implementation's discretion. Applications **should** only use operations defined for both types of file descriptors, unless they know via means external to Vulkan the type of the file descriptor, or are prepared to deal with the system-defined operation failures resulting from using the wrong type.

Some external semaphore handle types can only be shared within the same underlying physical device and/or the same driver version, as defined in the following table:

Table 76. External Semaphore Handle Types Compatibility

Handle type	VkPhysicalDeviceIDProperties::driverUUID	VkPhysicalDeviceIDProperties::deviceUUID
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_FD_BIT	Must match	Must match
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_BIT	Must match	Must match
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT	Must match	Must match
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_D3D12_FENCE_BIT	Must match	Must match
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SYNC_FD_BIT	No restriction	No restriction
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV	No restriction	No restriction

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkExternalSemaphoreHandleTypeFlags;
```

`VkExternalSemaphoreHandleTypeFlags` is a bitmask type for setting a mask of zero or more `VkExternalSemaphoreHandleTypeFlagBits`.

The `VkExternalSemaphoreProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalSemaphoreProperties {
    VkStructureType           sType;
    void*                     pNext;
    VkExternalSemaphoreHandleTypeFlags exportFromImportedHandleTypes;
    VkExternalSemaphoreHandleTypeFlags compatibleHandleTypes;
    VkExternalSemaphoreFeatureFlags externalSemaphoreFeatures;
} VkExternalSemaphoreProperties;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `exportFromImportedHandleTypes` is a bitmask of `VkExternalSemaphoreHandleTypeFlagBits` specifying which types of imported handle `handleType` **can** be exported from.
- `compatibleHandleTypes` is a bitmask of `VkExternalSemaphoreHandleTypeFlagBits` specifying handle types which **can** be specified at the same time as `handleType` when creating a semaphore.
- `externalSemaphoreFeatures` is a bitmask of `VkExternalSemaphoreFeatureFlagBits` describing the features of `handleType`.

If `handleType` is not supported by the implementation, then `VkExternalSemaphoreProperties::externalSemaphoreFeatures` will be zero.

Valid Usage (Implicit)

- `VUID-VkExternalSemaphoreProperties-sType-sType`
`sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_SEMAPHORE_PROPERTIES`
- `VUID-VkExternalSemaphoreProperties-pNext-pNext`
`pNext` **must** be `NULL`

Bits which **may** be set in `VkExternalSemaphoreProperties::externalSemaphoreFeatures`, specifying the features of an external semaphore handle type, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkExternalSemaphoreFeatureFlagBits {
    VK_EXTERNAL_SEMAPHORE_FEATURE_EXPORTABLE_BIT = 0x00000001,
    VK_EXTERNAL_SEMAPHORE_FEATURE_IMPORTABLE_BIT = 0x00000002,
} VkExternalSemaphoreFeatureFlagBits;
```

- `VK_EXTERNAL_SEMAPHORE_FEATURE_EXPORTABLE_BIT` specifies that handles of this type **can** be exported from Vulkan semaphore objects.
- `VK_EXTERNAL_SEMAPHORE_FEATURE_IMPORTABLE_BIT` specifies that handles of this type **can** be imported as Vulkan semaphore objects.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkExternalSemaphoreFeatureFlags;
```

`VkExternalSemaphoreFeatureFlags` is a bitmask type for setting a mask of zero or more `VkExternalSemaphoreFeatureFlagBits`.

36.4. Optional Fence Capabilities

Fences **may** support import and export of their `payload` to external handles. To query the external handle types supported by fences, call:

```
// Provided by VK_VERSION_1_1
void vkGetPhysicalDeviceExternalFenceProperties(
    VkPhysicalDevice          physicalDevice,
    const VkPhysicalDeviceExternalFenceInfo* pExternalFenceInfo,
    VkExternalFenceProperties* pExternalFenceProperties);
```

- `physicalDevice` is the physical device from which to query the fence capabilities.
- `pExternalFenceInfo` is a pointer to a `VkPhysicalDeviceExternalFenceInfo` structure describing the

parameters that would be consumed by [vkCreateFence](#).

- [pExternalFenceProperties](#) is a pointer to a [VkExternalFenceProperties](#) structure in which capabilities are returned.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceExternalFenceProperties-physicalDevice-parameter [physicalDevice](#) **must** be a valid [VkPhysicalDevice](#) handle
- VUID-vkGetPhysicalDeviceExternalFenceProperties-pExternalFenceInfo-parameter [pExternalFenceInfo](#) **must** be a valid pointer to a valid [VkPhysicalDeviceExternalFenceInfo](#) structure
- VUID-vkGetPhysicalDeviceExternalFenceProperties-pExternalFenceProperties-parameter [pExternalFenceProperties](#) **must** be a valid pointer to a [VkExternalFenceProperties](#) structure

The [VkPhysicalDeviceExternalFenceInfo](#) structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkPhysicalDeviceExternalFenceInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkExternalFenceHandleTypeFlagBits handleType;
} VkPhysicalDeviceExternalFenceInfo;
```

- [sType](#) is a [VkStructureType](#) value identifying this structure.
- [pNext](#) is [NULL](#) or a pointer to a structure extending this structure.
- [handleType](#) is a [VkExternalFenceHandleTypeFlagBits](#) value specifying an external fence handle type for which capabilities will be returned.

NOTE

Handles of type [VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT](#) generated by the implementation may represent either Linux Sync Files or Android Fences at the implementation's discretion. Applications **should** only use operations defined for both types of file descriptors, unless they know via means external to Vulkan the type of the file descriptor, or are prepared to deal with the system-defined operation failures resulting from using the wrong type.

Valid Usage (Implicit)

- VUID-VkPhysicalDeviceExternalFenceInfo-sType-sType [sType](#) **must** be [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FENCE_INFO](#)
- VUID-VkPhysicalDeviceExternalFenceInfo-pNext-pNext [pNext](#) **must** be [NULL](#)
- VUID-VkPhysicalDeviceExternalFenceInfo-handleType-parameter

`handleType` **must** be a valid `VkExternalFenceHandleTypeFlagBits` value

Bits which **may** be set in

- `VkPhysicalDeviceExternalFenceInfo::handleType`
- `VkExternalFenceProperties::exportFromImportedHandleTypes`
- `VkExternalFenceProperties::compatibleHandleTypes`

indicate external fence handle types, and are:

```
// Provided by VK_VERSION_1_1
typedef enum VkExternalFenceHandleTypeFlagBits {
    VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_FD_BIT = 0x00000001,
    VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT = 0x00000002,
    VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT = 0x00000004,
    VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT = 0x00000008,
    // Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
    VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV = 0x00000010,
    // Provided by VK_NV_external_sci_sync, VK_NV_external_sci_sync2
    VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV = 0x00000020,
} VkExternalFenceHandleTypeFlagBits;
```

- `VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_FD_BIT` specifies a POSIX file descriptor handle that has only limited valid usage outside of Vulkan and other compatible APIs. It **must** be compatible with the POSIX system calls `dup`, `dup2`, `close`, and the non-standard system call `dup3`. Additionally, it **must** be transportable over a socket using an `SCM_RIGHTS` control message. It owns a reference to the underlying synchronization primitive represented by its Vulkan fence object.
- `VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT` specifies an NT handle that has only limited valid usage outside of Vulkan and other compatible APIs. It **must** be compatible with the functions `DuplicateHandle`, `CloseHandle`, `CompareObjectHandles`, `GetHandleInformation`, and `SetHandleInformation`. It owns a reference to the underlying synchronization primitive represented by its Vulkan fence object.
- `VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT` specifies a global share handle that has only limited valid usage outside of Vulkan and other compatible APIs. It is not compatible with any native APIs. It does not own a reference to the underlying synchronization primitive represented by its Vulkan fence object, and will therefore become invalid when all Vulkan fence objects associated with it are destroyed.
- `VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT` specifies a POSIX file descriptor handle to a Linux Sync File or Android Fence. It can be used with any native API accepting a valid sync file or fence as input. It owns a reference to the underlying synchronization primitive associated with the file descriptor. Implementations which support importing this handle type **must** accept any type of sync or fence FD supported by the native system they are running on.
- `VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV` specifies a synchronization object (`NvSciSyncObj`) shareable across various hardware engines including the CPU and software (intra-process and inter-process) operating domains and perform signal and wait operations.

- `VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV` specifies a structure of `NvSciSyncFence` that is a snapshot of a synchronization object's underlying primitive and represents its possible state.

Some external fence handle types can only be shared within the same underlying physical device and/or the same driver version, as defined in the following table:

Table 77. External Fence Handle Types Compatibility

Handle type	VkPhysicalDeviceIDProperties::driverUUID	VkPhysicalDeviceIDProperties::deviceUUID
VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_FD_BIT	Must match	Must match
VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_BIT	Must match	Must match
VK_EXTERNAL_FENCE_HANDLE_TYPE_OPAQUE_WIN32_KMT_BIT	Must match	Must match
VK_EXTERNAL_FENCE_HANDLE_TYPE_SYNC_FD_BIT	No restriction	No restriction
VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV	Must match	Must match
VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV	Must match	Must match

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkExternalFenceHandleTypeFlags;
```

`VkExternalFenceHandleTypeFlags` is a bitmask type for setting a mask of zero or more `VkExternalFenceHandleTypeFlagBits`.

The `VkExternalFenceProperties` structure is defined as:

```
// Provided by VK_VERSION_1_1
typedef struct VkExternalFenceProperties {
    VkStructureType           sType;
    void*                     pNext;
    VkExternalFenceHandleTypeFlags exportFromImportedHandleTypes;
    VkExternalFenceHandleTypeFlags compatibleHandleTypes;
    VkExternalFenceFeatureFlags externalFenceFeatures;
} VkExternalFenceProperties;
```

- `exportFromImportedHandleTypes` is a bitmask of `VkExternalFenceHandleTypeFlagBits` indicating which types of imported handle `handleType` **can** be exported from.
- `compatibleHandleTypes` is a bitmask of `VkExternalFenceHandleTypeFlagBits` specifying handle types which **can** be specified at the same time as `handleType` when creating a fence.
- `externalFenceFeatures` is a bitmask of `VkExternalFenceFeatureFlagBits` indicating the features of `handleType`.

If `handleType` is not supported by the implementation, then `VkExternalFenceProperties::externalFenceFeatures` will be zero.

Valid Usage (Implicit)

- VUID-VkExternalFenceProperties-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_EXTERNAL_FENCE_PROPERTIES`
- VUID-VkExternalFenceProperties-pNext-pNext
`pNext` **must** be `NULL`

Bits which **may** be set in `VkExternalFenceProperties::externalFenceFeatures`, indicating features of a fence external handle type, are:

```
// Provided by VK_VERSION_1_1
typedef enum VkExternalFenceFeatureFlagBits {
    VK_EXTERNAL_FENCE_FEATURE_EXPORTABLE_BIT = 0x00000001,
    VK_EXTERNAL_FENCE_FEATURE_IMPORTABLE_BIT = 0x00000002,
} VkExternalFenceFeatureFlagBits;
```

- `VK_EXTERNAL_FENCE_FEATURE_EXPORTABLE_BIT` specifies handles of this type **can** be exported from Vulkan fence objects.
- `VK_EXTERNAL_FENCE_FEATURE_IMPORTABLE_BIT` specifies handles of this type **can** be imported to Vulkan fence objects.

```
// Provided by VK_VERSION_1_1
typedef VkFlags VkExternalFenceFeatureFlags;
```

`VkExternalFenceFeatureFlags` is a bitmask type for setting a mask of zero or more `VkExternalFenceFeatureFlagBits`.

36.5. Timestamp Calibration Capabilities

To query the set of time domains for which a physical device supports timestamp calibration, call:

```
// Provided by VK_KHR_calibrated_timestamps
VkResult vkGetPhysicalDeviceCalibrateableTimeDomainsKHR(
    VkPhysicalDevice          physicalDevice,
    uint32_t*                 pTimeDomainCount,
    VkTimeDomainKHR*          pTimeDomains);
```

- `physicalDevice` is the physical device from which to query the set of calibrateable time domains.
- `pTimeDomainCount` is a pointer to an integer related to the number of calibrateable time domains available or queried, as described below.
- `pTimeDomains` is either `NULL` or a pointer to an array of `VkTimeDomainKHR` values, indicating the supported calibrateable time domains.

If `pTimeDomains` is `NULL`, then the number of calibrateable time domains supported for the given `physicalDevice` is returned in `pTimeDomainCount`. Otherwise, `pTimeDomainCount` **must** point to a variable set by the application to the number of elements in the `pTimeDomains` array, and on return the variable is overwritten with the number of values actually written to `pTimeDomains`. If the value of `pTimeDomainCount` is less than the number of calibrateable time domains supported, at most `pTimeDomainCount` values will be written to `pTimeDomains`, and `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available time domains were returned.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetPhysicalDeviceCalibrateableTimeDomainsKHR` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceCalibrateableTimeDomainsKHR-physicalDevice-parameter `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- VUID-vkGetPhysicalDeviceCalibrateableTimeDomainsKHR-pTimeDomainCount-parameter `pTimeDomainCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetPhysicalDeviceCalibrateableTimeDomainsKHR-pTimeDomains-parameter
If the value referenced by `pTimeDomainCount` is not `0`, and `pTimeDomains` is not `NULL`, `pTimeDomains` **must** be a valid pointer to an array of `pTimeDomainCount` `VkTimeDomainKHR` values

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

36.6. Object Refresh Capabilities

To query the set of object types that require periodic refreshing, call:

```
// Provided by VK_KHR_object_refresh
VkResult vkGetPhysicalDeviceRefreshableObjectTypesKHR(
    VkPhysicalDevice          physicalDevice,
    uint32_t*                 pRefreshableObjectTypeCount,
    VkObjectType*             pRefreshableObjectTypes);
```

- **physicalDevice** is the physical device from which to query the set of refreshable object types.
- **pRefreshableObjectTypeCount** is a pointer to an integer related to the number of refreshable object types available or queried, as described below.
- **pRefreshableObjectTypes** is either **NULL** or a pointer to an array of **VkObjectType** values, indicating the supported refreshable object types.

If **pRefreshableObjectTypes** is **NULL**, then the number of refreshable object types supported for the given **physicalDevice** is returned in **pRefreshableObjectTypeCount**. Otherwise, **pRefreshableObjectTypeCount** **must** point to a variable set by the application to the number of elements in the **pRefreshableObjectTypes** array, and on return the variable is overwritten with the number of object types actually written to **pRefreshableObjectTypes**. If the value of **pRefreshableObjectTypeCount** is less than the number of refreshable object types supported, at most **pRefreshableObjectTypeCount** object types will be written, and **VK_INCOMPLETE** will be returned instead of **VK_SUCCESS**, to indicate that not all the available object types were returned.

Valid Usage (Implicit)

- VUID-vkGetPhysicalDeviceRefreshableObjectTypesKHR-physicalDevice-parameter **physicalDevice** **must** be a valid **VkPhysicalDevice** handle
- VUID-vkGetPhysicalDeviceRefreshableObjectTypesKHR-pRefreshableObjectTypeCount-parameter **pRefreshableObjectTypeCount** **must** be a valid pointer to a **uint32_t** value
- VUID-vkGetPhysicalDeviceRefreshableObjectTypesKHR-pRefreshableObjectTypes-parameter

If the value referenced by **pRefreshableObjectTypeCount** is not **0**, and **pRefreshableObjectTypes** is not **NULL**, **pRefreshableObjectTypes** **must** be a valid pointer to an array of **pRefreshableObjectTypeCount** **VkObjectType** values

Return Codes

Success

- **VK_INCOMPLETE**
- **VK_SUCCESS**

Failure

- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

Chapter 37. Debugging

To aid developers in tracking down errors in the application's use of Vulkan, particularly in combination with an external debugger or profiler, *debugging extensions* may be available.

The [VkObjectType](#) enumeration defines values, each of which corresponds to a specific Vulkan handle type. These values **can** be used to associate debug information with a particular type of object through one or more extensions.

```

// Provided by VK_VERSION_1_0
typedef enum VkObjectType {
    VK_OBJECT_TYPE_UNKNOWN = 0,
    VK_OBJECT_TYPE_INSTANCE = 1,
    VK_OBJECT_TYPE_PHYSICAL_DEVICE = 2,
    VK_OBJECT_TYPE_DEVICE = 3,
    VK_OBJECT_TYPE_QUEUE = 4,
    VK_OBJECT_TYPE_SEMAPHORE = 5,
    VK_OBJECT_TYPE_COMMAND_BUFFER = 6,
    VK_OBJECT_TYPE_FENCE = 7,
    VK_OBJECT_TYPE_DEVICE_MEMORY = 8,
    VK_OBJECT_TYPE_BUFFER = 9,
    VK_OBJECT_TYPE_IMAGE = 10,
    VK_OBJECT_TYPE_EVENT = 11,
    VK_OBJECT_TYPE_QUERY_POOL = 12,
    VK_OBJECT_TYPE_BUFFER_VIEW = 13,
    VK_OBJECT_TYPE_IMAGE_VIEW = 14,
    VK_OBJECT_TYPE_SHADER_MODULE = 15,
    VK_OBJECT_TYPE_PIPELINE_CACHE = 16,
    VK_OBJECT_TYPE_PIPELINE_LAYOUT = 17,
    VK_OBJECT_TYPE_RENDER_PASS = 18,
    VK_OBJECT_TYPE_PIPELINE = 19,
    VK_OBJECT_TYPE_DESCRIPTOR_SET_LAYOUT = 20,
    VK_OBJECT_TYPE_SAMPLER = 21,
    VK_OBJECT_TYPE_DESCRIPTOR_POOL = 22,
    VK_OBJECT_TYPE_DESCRIPTOR_SET = 23,
    VK_OBJECT_TYPE_FRAMEBUFFER = 24,
    VK_OBJECT_TYPE_COMMAND_POOL = 25,
    // Provided by VK_VERSION_1_1
    VK_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION = 1000156000,
    // Provided by VK_KHR_surface
    VK_OBJECT_TYPE_SURFACE_KHR = 1000000000,
    // Provided by VK_KHR_swapchain
    VK_OBJECT_TYPE_SWAPCHAIN_KHR = 1000001000,
    // Provided by VK_KHR_display
    VK_OBJECT_TYPE_DISPLAY_KHR = 1000002000,
    // Provided by VK_KHR_display
    VK_OBJECT_TYPE_DISPLAY_MODE_KHR = 1000002001,
    // Provided by VK_EXT_debug_utils
    VK_OBJECT_TYPE_DEBUG_UTILS_MESSENGER_EXT = 1000128000,
    // Provided by VK_NV_external_sci_sync2
    VK_OBJECT_TYPE_SEMAPHORE_SCI_SYNC_POOL_NV = 1000489000,
} VkObjectType;

```

Table 78. **VkObjectType** and Vulkan Handle Relationship

VkObjectType	Vulkan Handle Type
VK_OBJECT_TYPE_UNKNOWN	Unknown/Undefined Handle
VK_OBJECT_TYPE_INSTANCE	VkInstance

VkObjectType	Vulkan Handle Type
VK_OBJECT_TYPE_PHYSICAL_DEVICE	VkPhysicalDevice
VK_OBJECT_TYPE_DEVICE	VkDevice
VK_OBJECT_TYPE_QUEUE	VkQueue
VK_OBJECT_TYPE_SEMAPHORE	VkSemaphore
VK_OBJECT_TYPE_COMMAND_BUFFER	VkCommandBuffer
VK_OBJECT_TYPE_FENCE	VkFence
VK_OBJECT_TYPE_DEVICE_MEMORY	VkDeviceMemory
VK_OBJECT_TYPE_BUFFER	VkBuffer
VK_OBJECT_TYPE_IMAGE	VkImage
VK_OBJECT_TYPE_EVENT	VkEvent
VK_OBJECT_TYPE_QUERY_POOL	VkQueryPool
VK_OBJECT_TYPE_BUFFER_VIEW	VkBufferView
VK_OBJECT_TYPE_IMAGE_VIEW	VkImageView
VK_OBJECT_TYPE_PIPELINE_CACHE	VkPipelineCache
VK_OBJECT_TYPE_PIPELINE_LAYOUT	VkPipelineLayout
VK_OBJECT_TYPE_RENDER_PASS	VkRenderPass
VK_OBJECT_TYPE_PIPELINE	VkPipeline
VK_OBJECT_TYPE_DESCRIPTOR_SET_LAYOUT	VkDescriptorSetLayout
VK_OBJECT_TYPE_SAMPLER	VkSampler
VK_OBJECT_TYPE_DESCRIPTOR_POOL	VkDescriptorPool
VK_OBJECT_TYPE_DESCRIPTOR_SET	VkDescriptorSet
VK_OBJECT_TYPE_FRAMEBUFFER	VkFramebuffer
VK_OBJECT_TYPE_COMMAND_POOL	VkCommandPool
VK_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION	VkSamplerYcbcrConversion
VK_OBJECT_TYPE_SURFACE_KHR	VkSurfaceKHR
VK_OBJECT_TYPE_SWAPCHAIN_KHR	VkSwapchainKHR
VK_OBJECT_TYPE_DISPLAY_KHR	VkDisplayKHR
VK_OBJECT_TYPE_DISPLAY_MODE_KHR	VkDisplayModeKHR
VK_OBJECT_TYPE_DEBUG_UTILS_MESSENGER_EXT	VkDebugUtilsMessengerEXT

If this Specification was generated with any such extensions included, they will be described in the remainder of this chapter.

37.1. Debug Utilities

Vulkan provides flexible debugging utilities for debugging an application.

The [Object Debug Annotation](#) section describes how to associate either a name or binary data with a specific Vulkan object.

The [Queue Labels](#) section describes how to annotate and group the work submitted to a queue.

The [Command Buffer Labels](#) section describes how to associate logical elements of the scene with commands in a [VkCommandBuffer](#).

The [Debug Messengers](#) section describes how to create debug messenger objects associated with an application supplied callback to capture debug messages from a variety of Vulkan components.

37.1.1. Object Debug Annotation

It can be useful for an application to provide its own content relative to a specific Vulkan object.

The following commands allow application developers to associate application-defined information with Vulkan objects. These commands are device-level commands but they **may** reference instance-level objects (such as [VkInstance](#)) and physical device-level objects (such as [VkPhysicalDevice](#)) with a few restrictions: * The data for the corresponding object **may** still be available after the [VkDevice](#) used in the corresponding API call to set it is destroyed, but access to this data is not guaranteed and should be avoided. * Subsequent calls to change the data of the same object across multiple [VkDevice](#) objects, **may** result in the data being changed to the most recent version for all [VkDevice](#) objects and not just the [VkDevice](#) used in the most recent API call.

Object Naming

An object can be given an application-defined name by calling:

```
// Provided by VK_EXT_debug_utils
VkResult vkSetDebugUtilsObjectNameEXT(
    VkDevice device,
    const VkDebugUtilsObjectNameInfoEXT* pNameInfo);
```

- [device](#) is the device that is associated with the named object passed in via [objectHandle](#).
- [pNameInfo](#) is a pointer to a [VkDebugUtilsObjectNameInfoEXT](#) structure specifying parameters of the name to set on the object.

Valid Usage

- VUID-vkSetDebugUtilsObjectNameEXT-pNameInfo-02587
[pNameInfo->objectType](#) **must** not be [VK_OBJECT_TYPE_UNKNOWN](#)
- VUID-vkSetDebugUtilsObjectNameEXT-pNameInfo-02588
[pNameInfo->objectHandle](#) **must** not be [VK_NULL_HANDLE](#)

- VUID-vkSetDebugUtilsObjectNameEXT-pNameInfo-07872
If `pNameInfo->objectHandle` is the valid handle of an instance-level object, the `VkDevice` identified by `device` **must** be a descendent of the same `VkInstance` as the object identified by `pNameInfo->objectHandle`
- VUID-vkSetDebugUtilsObjectNameEXT-pNameInfo-07873
If `pNameInfo->objectHandle` is the valid handle of a physical-device-level object, the `VkDevice` identified by `device` **must** be a descendant of the same `VkPhysicalDevice` as the object identified by `pNameInfo->objectHandle`
- VUID-vkSetDebugUtilsObjectNameEXT-pNameInfo-07874
If `pNameInfo->objectHandle` is the valid handle of a device-level object, that object **must** be a descendent of the `VkDevice` identified by `device`

Valid Usage (Implicit)

- VUID-vkSetDebugUtilsObjectNameEXT-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkSetDebugUtilsObjectNameEXT-pNameInfo-parameter
`pNameInfo` **must** be a valid pointer to a valid `VkDebugUtilsObjectNameInfoEXT` structure

Host Synchronization

- Host access to `pNameInfo->objectHandle` **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkDebugUtilsObjectNameInfoEXT` structure is defined as:

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsObjectNameInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkObjectType        objectType;
    uint64_t           objectHandle;
    const char*         pObjectName;
} VkDebugUtilsObjectNameInfoEXT;
```

- **sType** is a [VkStructureType](#) value identifying this structure.
- **pNext** is **NULL** or a pointer to a structure extending this structure.
- **objectType** is a [VkObjectType](#) specifying the type of the object to be named.
- **objectHandle** is the object to be named.
- **pObjectName** is either **NULL** or a null-terminated UTF-8 string specifying the name to apply to **objectHandle**.

Applications **may** change the name associated with an object simply by calling **vkSetDebugUtilsObjectNameEXT** again with a new string. If **pObjectName** is either **NULL** or an empty string, then any previously set name is removed.

Valid Usage

- VUID-VkDebugUtilsObjectNameInfoEXT-objectType-02589
If **objectType** is **VK_OBJECT_TYPE_UNKNOWN**, **objectHandle** **must** not be [VK_NULL_HANDLE](#)
- VUID-VkDebugUtilsObjectNameInfoEXT-objectType-02590
If **objectType** is not **VK_OBJECT_TYPE_UNKNOWN**, **objectHandle** **must** be [VK_NULL_HANDLE](#) or a valid Vulkan handle of the type associated with **objectType** as defined in the [VkObjectType](#) and [Vulkan Handle Relationship](#) table

Valid Usage (Implicit)

- VUID-VkDebugUtilsObjectNameInfoEXT-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT**
- VUID-VkDebugUtilsObjectNameInfoEXT-objectType-parameter
objectType **must** be a valid [VkObjectType](#) value
- VUID-VkDebugUtilsObjectNameInfoEXT-pObjectName-parameter
If **pObjectName** is not **NULL**, **pObjectName** **must** be a null-terminated UTF-8 string

Object Data Association

In addition to setting a name for an object, debugging and validation layers **may** have uses for additional binary data on a per-object basis that have no other place in the Vulkan API.

For example, a `VkShaderModule` could have additional debugging data attached to it to aid in offline shader tracing.

Additional data can be attached to an object by calling `vkSetDebugUtilsObjectTagEXT` as defined below.

```
// Provided by VK_EXT_debug_utils
VkResult vkSetDebugUtilsObjectTagEXT(
    VkDevice device,
    const VkDebugUtilsObjectTagInfoEXT* pTagInfo);
```

- `device` is the device that created the object.
- `pTagInfo` is a pointer to a `VkDebugUtilsObjectTagInfoEXT` structure specifying parameters of the tag to attach to the object.

Valid Usage

- VUID-vkSetDebugUtilsObjectTagEXT-pNameInfo-07875
If `pNameInfo->objectHandle` is the valid handle of an instance-level object, the `VkDevice` identified by `device` **must** be a descendent of the same `VkInstance` as the object identified by `pNameInfo->objectHandle`
- VUID-vkSetDebugUtilsObjectTagEXT-pNameInfo-07876
If `pNameInfo->objectHandle` is the valid handle of a physical-device-level object, the `VkDevice` identified by `device` **must** be a descendant of the same `VkPhysicalDevice` as the object identified by `pNameInfo->objectHandle`
- VUID-vkSetDebugUtilsObjectTagEXT-pNameInfo-07877
If `pNameInfo->objectHandle` is the valid handle of a device-level object, that object **must** be a descendent of the `VkDevice` identified by `device`

Valid Usage (Implicit)

- VUID-vkSetDebugUtilsObjectTagEXT-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkSetDebugUtilsObjectTagEXT-pTagInfo-parameter
`pTagInfo` **must** be a valid pointer to a valid `VkDebugUtilsObjectTagInfoEXT` structure

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

The `VkDebugUtilsObjectTagInfoEXT` structure is defined as:

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsObjectTagInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkObjectType        objectType;
    uint64_t            objectHandle;
    uint64_t            tagName;
    size_t              tagSize;
    const void*        pTag;
} VkDebugUtilsObjectTagInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `objectType` is a `VkObjectType` specifying the type of the object to be named.
- `objectHandle` is the object to be tagged.
- `tagName` is a numerical identifier of the tag.
- `tagSize` is the number of bytes of data to attach to the object.
- `pTag` is a pointer to an array of `tagSize` bytes containing the data to be associated with the object.

The `tagName` parameter gives a name or identifier to the type of data being tagged. This can be used by debugging layers to easily filter for only data that can be used by that implementation.

Valid Usage

- VUID-VkDebugUtilsObjectTagInfoEXT-objectType-01908
`objectType` **must** not be `VK_OBJECT_TYPE_UNKNOWN`
- VUID-VkDebugUtilsObjectTagInfoEXT-objectHandle-01910
`objectHandle` **must** be a valid Vulkan handle of the type associated with `objectType` as defined in the `VkObjectType` and `Vulkan Handle Relationship` table

Valid Usage (Implicit)

- VUID-VkDebugUtilsObjectTagInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_TAG_INFO_EXT`
- VUID-VkDebugUtilsObjectTagInfoEXT-pNext-pNext
`pNext` **must** be `NULL`

- VUID-VkDebugUtilsObjectTagInfoEXT-objectType-parameter **objectType** must be a valid [VkObjectType](#) value
- VUID-VkDebugUtilsObjectTagInfoEXT-pTag-parameter **pTag** must be a valid pointer to an array of **tagSize** bytes
- VUID-VkDebugUtilsObjectTagInfoEXT-tagSize-arraylength **tagSize** must be greater than 0

Host Synchronization

- Host access to **objectHandle** must be externally synchronized

37.1.2. Queue Labels

All Vulkan work is submitted using queues. It is possible for an application to use multiple queues, each containing multiple command buffers, when performing work. It can be useful to identify which queue, or even where in a queue, something has occurred.

To begin identifying a region using a debug label inside a queue, you may use the [vkQueueBeginDebugUtilsLabelEXT](#) command.

Then, when the region of interest has passed, you may end the label region using [vkQueueEndDebugUtilsLabelEXT](#).

Additionally, a single debug label may be inserted at any time using [vkQueueInsertDebugUtilsLabelEXT](#).

A queue debug label region is opened by calling:

```
// Provided by VK_EXT_debug_utils
void vkQueueBeginDebugUtilsLabelEXT(
    VkQueue queue,
    const VkDebugUtilsLabelEXT* pLabelInfo);
```

- **queue** is the queue in which to start a debug label region.
- **pLabelInfo** is a pointer to a [VkDebugUtilsLabelEXT](#) structure specifying parameters of the label region to open.

Valid Usage (Implicit)

- VUID-vkQueueBeginDebugUtilsLabelEXT-queue-parameter **queue** must be a valid [VkQueue](#) handle
- VUID-vkQueueBeginDebugUtilsLabelEXT-pLabelInfo-parameter **pLabelInfo** must be a valid pointer to a valid [VkDebugUtilsLabelEXT](#) structure

Host Synchronization

- Host access to **queue** **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
-	-	Any	-

The `VkDebugUtilsLabelEXT` structure is defined as:

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsLabelEXT {
    VkStructureType    sType;
    const void*        pNext;
    const char*        pLabelName;
    float              color[4];
} VkDebugUtilsLabelEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `pLabelName` is a pointer to a null-terminated UTF-8 string containing the name of the label.
- `color` is an optional RGBA color value that can be associated with the label. A particular implementation **may** choose to ignore this color value. The values contain RGBA values in order, in the range 0.0 to 1.0. If all elements in `color` are 0.0, then it is ignored.

Valid Usage (Implicit)

- VUID-VkDebugUtilsLabelEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT`
- VUID-VkDebugUtilsLabelEXT-pNext-pNext
`pNext` **must** be `NULL`
- VUID-VkDebugUtilsLabelEXT-pLabelName-parameter
`pLabelName` **must** be a null-terminated UTF-8 string

A queue debug label region is closed by calling:

```
// Provided by VK_EXT_debug_utils
void vkQueueEndDebugUtilsLabelEXT(
    VkQueue                                     queue);
```

- **queue** is the queue in which a debug label region should be closed.

The calls to `vkQueueBeginDebugUtilsLabelEXT` and `vkQueueEndDebugUtilsLabelEXT` **must** be matched and balanced.

Valid Usage

- VUID-vkQueueEndDebugUtilsLabelEXT-None-01911
There **must** be an outstanding `vkQueueBeginDebugUtilsLabelEXT` command prior to the `vkQueueEndDebugUtilsLabelEXT` on the queue

Valid Usage (Implicit)

- VUID-vkQueueEndDebugUtilsLabelEXT-queue-parameter
queue **must** be a valid `VkQueue` handle

Host Synchronization

- Host access to **queue** **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
-	-	Any	-

A single label can be inserted into a queue by calling:

```
// Provided by VK_EXT_debug_utils
void vkQueueInsertDebugUtilsLabelEXT(
    VkQueue                                     queue,
    const VkDebugUtilsLabelEXT*                pLabelInfo);
```

- **queue** is the queue into which a debug label will be inserted.
- **pLabelInfo** is a pointer to a `VkDebugUtilsLabelEXT` structure specifying parameters of the label to insert.

Valid Usage (Implicit)

- VUID-vkQueueInsertDebugUtilsLabelEXT-queue-parameter
queue **must** be a valid [VkQueue](#) handle
- VUID-vkQueueInsertDebugUtilsLabelEXT-pLabelInfo-parameter
pLabelInfo **must** be a valid pointer to a valid [VkDebugUtilsLabelEXT](#) structure

Host Synchronization

- Host access to **queue** **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
-	-	Any	-

37.1.3. Command Buffer Labels

Typical Vulkan applications will submit many command buffers in each frame, with each command buffer containing a large number of individual commands. Being able to logically annotate regions of command buffers that belong together as well as hierarchically subdivide the frame is important to a developer's ability to navigate the commands viewed holistically.

To identify the beginning of a debug label region in a command buffer, [vkCmdBeginDebugUtilsLabelEXT](#) **can** be used as defined below.

To indicate the end of a debug label region in a command buffer, [vkCmdEndDebugUtilsLabelEXT](#) **can** be used.

To insert a single command buffer debug label inside of a command buffer, [vkCmdInsertDebugUtilsLabelEXT](#) **can** be used as defined below.

A command buffer debug label region can be opened by calling:

```
// Provided by VK_EXT_debug_utils
void vkCmdBeginDebugUtilsLabelEXT(
    VkCommandBuffer                commandBuffer,
    const VkDebugUtilsLabelEXT*    pLabelInfo);
```

- **commandBuffer** is the command buffer into which the command is recorded.
- **pLabelInfo** is a pointer to a [VkDebugUtilsLabelEXT](#) structure specifying parameters of the label region to open.

Valid Usage (Implicit)

- VUID-vkCmdBeginDebugUtilsLabelEXT-commandBuffer-parameter `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdBeginDebugUtilsLabelEXT-pLabelInfo-parameter `pLabelInfo` **must** be a valid pointer to a valid `VkDebugUtilsLabelEXT` structure
- VUID-vkCmdBeginDebugUtilsLabelEXT-commandBuffer-recording `commandBuffer` **must** be in the `recording` state
- VUID-vkCmdBeginDebugUtilsLabelEXT-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action State

A command buffer label region can be closed by calling:

```
// Provided by VK_EXT_debug_utils
void vkCmdEndDebugUtilsLabelEXT(
    VkCommandBuffer                commandBuffer);
```

- `commandBuffer` is the command buffer into which the command is recorded.

An application **may** open a debug label region in one command buffer and close it in another, or otherwise split debug label regions across multiple command buffers or multiple queue submissions. When viewed from the linear series of submissions to a single queue, the calls to `vkCmdBeginDebugUtilsLabelEXT` and `vkCmdEndDebugUtilsLabelEXT` **must** be matched and balanced.

There **can** be problems reporting command buffer debug labels during the recording process because command buffers **may** be recorded out of sequence with the resulting execution order. Since the recording order **may** be different, a solitary command buffer **may** have an inconsistent view of the debug label regions by itself. Therefore, if an issue occurs during the recording of a command buffer, and the environment requires returning debug labels, the implementation **may** return only those labels it is aware of. This is true even if the implementation is aware of only the debug labels within the command buffer being actively recorded.

Valid Usage

- VUID-vkCmdEndDebugUtilsLabelEXT-commandBuffer-01912
There **must** be an outstanding `vkCmdBeginDebugUtilsLabelEXT` command prior to the `vkCmdEndDebugUtilsLabelEXT` on the queue that `commandBuffer` is submitted to
- VUID-vkCmdEndDebugUtilsLabelEXT-commandBuffer-01913
If `commandBuffer` is a secondary command buffer, there **must** be an outstanding `vkCmdBeginDebugUtilsLabelEXT` command recorded to `commandBuffer` that has not previously been ended by a call to `vkCmdEndDebugUtilsLabelEXT`

Valid Usage (Implicit)

- VUID-vkCmdEndDebugUtilsLabelEXT-commandBuffer-parameter
`commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdEndDebugUtilsLabelEXT-commandBuffer-recording
`commandBuffer` **must** be in the `recording state`
- VUID-vkCmdEndDebugUtilsLabelEXT-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action State

A single debug label can be inserted into a command buffer by calling:

```
// Provided by VK_EXT_debug_utils
void vkCmdInsertDebugUtilsLabelEXT(
    VkCommandBuffer          commandBuffer,
    const VkDebugUtilsLabelEXT* pLabelInfo);
```

- `commandBuffer` is the command buffer into which the command is recorded.
- `pLabelInfo` is a pointer to a `VkDebugUtilsLabelEXT` structure specifying parameters of the label to insert.

Valid Usage (Implicit)

- VUID-vkCmdInsertDebugUtilsLabelEXT-commandBuffer-parameter `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- VUID-vkCmdInsertDebugUtilsLabelEXT-pLabelInfo-parameter `pLabelInfo` **must** be a valid pointer to a valid `VkDebugUtilsLabelEXT` structure
- VUID-vkCmdInsertDebugUtilsLabelEXT-commandBuffer-recording `commandBuffer` **must** be in the `recording state`
- VUID-vkCmdInsertDebugUtilsLabelEXT-commandBuffer-cmdpool
The `VkCommandPool` that `commandBuffer` was allocated from **must** support `VK_QUEUE_COMPUTE_BIT`, `VK_QUEUE_GRAPHICS_BIT`, or `VK_QUEUE_TRANSFER_BIT` operations

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized
- Host access to the `VkCommandPool` that `commandBuffer` was allocated from **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Command Type
Primary Secondary	Both	VK_QUEUE_COMPUTE_BIT VK_QUEUE_GRAPHICS_BIT VK_QUEUE_TRANSFER_BIT	Action

37.1.4. Debug Messengers

Vulkan allows an application to register multiple callbacks with any Vulkan component wishing to report debug information. Some callbacks may log the information to a file, others may cause a debug break point or other application-defined behavior. A primary producer of callback messages are the validation layers. An application **can** register callbacks even when no validation layers are enabled, but they will only be called for the Vulkan loader and, if implemented, other layer and driver events.

A `VkDebugUtilsMessengerEXT` is a messenger object which handles passing along debug messages to a provided debug callback.

```
// Provided by VK_EXT_debug_utils
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDebugUtilsMessengerEXT)
```

The debug messenger will provide detailed feedback on the application's use of Vulkan when events of interest occur. When an event of interest does occur, the debug messenger will submit a debug message to the debug callback that was provided during its creation. Additionally, the debug messenger is responsible with filtering out debug messages that the callback is not interested in and will only provide desired debug messages.

A debug messenger triggers a debug callback with a debug message when an event of interest occurs. To create a debug messenger which will trigger a debug callback, call:

```
// Provided by VK_EXT_debug_utils
VkResult vkCreateDebugUtilsMessengerEXT(
    VkInstance          instance,
    const VkDebugUtilsMessengerCreateInfoEXT* pCreateInfo,
    const VkAllocationCallbacks* pAllocator,
    VkDebugUtilsMessengerEXT* pMessenger);
```

- `instance` is the instance the messenger will be used with.
- `pCreateInfo` is a pointer to a `VkDebugUtilsMessengerCreateInfoEXT` structure containing the

callback pointer, as well as defining conditions under which this messenger will trigger the callback.

- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.
- **pMessenger** is a pointer to a [VkDebugUtilsMessengerEXT](#) handle in which the created object is returned.

Valid Usage (Implicit)

- VUID-vkCreateDebugUtilsMessengerEXT-instance-parameter **instance** **must** be a valid [VkInstance](#) handle
- VUID-vkCreateDebugUtilsMessengerEXT-pCreateInfo-parameter **pCreateInfo** **must** be a valid pointer to a valid [VkDebugUtilsMessengerCreateInfoEXT](#) structure
- VUID-vkCreateDebugUtilsMessengerEXT-pAllocator-null **pAllocator** **must** be **NULL**
- VUID-vkCreateDebugUtilsMessengerEXT-pMessenger-parameter **pMessenger** **must** be a valid pointer to a [VkDebugUtilsMessengerEXT](#) handle

Return Codes

Success

- **VK_SUCCESS**

Failure

- **VK_ERROR_OUT_OF_HOST_MEMORY**
- **VK_ERROR_UNKNOWN**
- **VK_ERROR_VALIDATION_FAILED**

The application **must** ensure that [vkCreateDebugUtilsMessengerEXT](#) is not executed in parallel with any Vulkan command that is also called with **instance** or child of **instance** as the dispatchable argument.

The definition of [VkDebugUtilsMessengerCreateInfoEXT](#) is:

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsMessengerCreateInfoEXT {
    VkStructureType           sType;
    const void*               pNext;
    VkDebugUtilsMessengerCreateFlagsEXT flags;
    VkDebugUtilsMessageSeverityFlagsEXT messageSeverity;
    VkDebugUtilsMessageTypeFlagsEXT messageType;
    PFN_vkDebugUtilsMessengerCallbackEXT pfnUserCallback;
    void*                     pUserData;
} VkDebugUtilsMessengerCreateInfoEXT;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is `0` and is reserved for future use.
- `messageSeverity` is a bitmask of `VkDebugUtilsMessageSeverityFlagBitsEXT` specifying which severity of event(s) will cause this callback to be called.
- `messageType` is a bitmask of `VkDebugUtilsMessageTypeFlagBitsEXT` specifying which type of event(s) will cause this callback to be called.
- `pfnUserCallback` is the application callback function to call.
- `pUserData` is user data to be passed to the callback.

For each `VkDebugUtilsMessengerEXT` that is created the `VkDebugUtilsMessengerCreateInfoEXT::messageSeverity` and `VkDebugUtilsMessengerCreateInfoEXT::messageType` determine when that `VkDebugUtilsMessengerCreateInfoEXT::pfnUserCallback` is called. The process to determine if the user's `pfnUserCallback` is triggered when an event occurs is as follows:

1. The implementation will perform a bitwise AND of the event's `VkDebugUtilsMessageSeverityFlagBitsEXT` with the `messageSeverity` provided during creation of the `VkDebugUtilsMessengerEXT` object.
 - a. If the value is 0, the message is skipped.
2. The implementation will perform bitwise AND of the event's `VkDebugUtilsMessageTypeFlagBitsEXT` with the `messageType` provided during the creation of the `VkDebugUtilsMessengerEXT` object.
 - a. If the value is 0, the message is skipped.
3. The callback will trigger a debug message for the current event

The callback will come directly from the component that detected the event, unless some other layer intercepts the calls for its own purposes (filter them in a different way, log to a system error log, etc.).

An application **can** receive multiple callbacks if multiple `VkDebugUtilsMessengerEXT` objects are created. A callback will always be executed in the same thread as the originating Vulkan call.

A callback **can** be called from multiple threads simultaneously (if the application is making Vulkan

calls from multiple threads).

Valid Usage (Implicit)

- VUID-VkDebugUtilsMessengerCreateInfoEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT`
- VUID-VkDebugUtilsMessengerCreateInfoEXT-flags-zerobitmask
`flags` **must** be `0`
- VUID-VkDebugUtilsMessengerCreateInfoEXT-messageSeverity-parameter
`messageSeverity` **must** be a valid combination of `VkDebugUtilsMessageSeverityFlagBitsEXT` values
- VUID-VkDebugUtilsMessengerCreateInfoEXT-messageSeverity-requiredbitmask
`messageSeverity` **must** not be `0`
- VUID-VkDebugUtilsMessengerCreateInfoEXT-messageType-parameter
`messageType` **must** be a valid combination of `VkDebugUtilsMessageTypeFlagBitsEXT` values
- VUID-VkDebugUtilsMessengerCreateInfoEXT-messageType-requiredbitmask
`messageType` **must** not be `0`
- VUID-VkDebugUtilsMessengerCreateInfoEXT-pfnUserCallback-parameter
`pfnUserCallback` **must** be a valid `PFN_vkDebugUtilsMessengerCallbackEXT` value

```
// Provided by VK_EXT_debug_utils
typedef VkFlags VkDebugUtilsMessengerCreateFlagsEXT;
```

`VkDebugUtilsMessengerCreateFlagsEXT` is a bitmask type for setting a mask, but is currently reserved for future use.

Bits which **can** be set in `VkDebugUtilsMessengerCreateInfoEXT::messageSeverity`, specifying event severities which cause a debug messenger to call the callback, are:

```
// Provided by VK_EXT_debug_utils
typedef enum VkDebugUtilsMessageSeverityFlagBitsEXT {
    VK_DEBUG_UTILS_MESSAGE_SEVERITY_VERBOSE_BIT_EXT = 0x00000001,
    VK_DEBUG_UTILS_MESSAGE_SEVERITY_INFO_BIT_EXT = 0x00000010,
    VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT = 0x00000100,
    VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT = 0x00001000,
} VkDebugUtilsMessageSeverityFlagBitsEXT;
```

- `VK_DEBUG_UTILS_MESSAGE_SEVERITY_VERBOSE_BIT_EXT` specifies the most verbose output indicating all diagnostic messages from the Vulkan loader, layers, and drivers should be captured.
- `VK_DEBUG_UTILS_MESSAGE_SEVERITY_INFO_BIT_EXT` specifies an informational message such as resource details that may be handy when debugging an application.
- `VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT` specifies use of Vulkan that **may** expose an

application bug. Such cases may not be immediately harmful, such as a fragment shader outputting to a location with no attachment. Other cases **may** point to behavior that is almost certainly bad when unintended such as using an image whose memory has not been filled. In general if you see a warning but you know that the behavior is intended/desired, then simply ignore the warning.

- `VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT` specifies that the application has violated a valid usage condition of the specification.

The values of `VkDebugUtilsMessageSeverityFlagBitsEXT` are sorted based on severity. The higher the flag value, the more severe the message. This allows for simple boolean operation comparisons when looking at `VkDebugUtilsMessageSeverityFlagBitsEXT` values.

For example:

NOTE

```
if (messageSeverity >=
VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT) {
    // Do something for warnings and errors
}
```

In addition, space has been left between the enums to allow for later addition of new severities in between the existing values.

```
// Provided by VK_EXT_debug_utils
typedef VkFlags VkDebugUtilsMessageSeverityFlagsEXT;
```

`VkDebugUtilsMessageSeverityFlagsEXT` is a bitmask type for setting a mask of zero or more `VkDebugUtilsMessageSeverityFlagBitsEXT`.

Bits which **can** be set in `VkDebugUtilsMessengerCreateInfoEXT::messageType`, specifying event types which cause a debug messenger to call the callback, are:

```
// Provided by VK_EXT_debug_utils
typedef enum VkDebugUtilsMessageTypeFlagBitsEXT {
    VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT = 0x00000001,
    VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT = 0x00000002,
    VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT = 0x00000004,
} VkDebugUtilsMessageTypeFlagBitsEXT;
```

- `VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT` specifies that some general event has occurred. This is typically a non-specification, non-performance event.
- `VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT` specifies that something has occurred during validation against the Vulkan specification that may indicate invalid behavior.
- `VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT` specifies a potentially non-optimal use of Vulkan, e.g. using `vkCmdClearColorImage` when setting `VkAttachmentDescription::loadOp` to

`VK_ATTACHMENT_LOAD_OP_CLEAR` would have worked.

```
// Provided by VK_EXT_debug_utils
typedef VkFlags VkDebugUtilsMessageTypeFlagsEXT;
```

`VkDebugUtilsMessageTypeFlagsEXT` is a bitmask type for setting a mask of zero or more `VkDebugUtilsMessageTypeFlagBitsEXT`.

The prototype for the `VkDebugUtilsMessengerCreateInfoEXT::pfnUserCallback` function implemented by the application is:

```
// Provided by VK_EXT_debug_utils
typedef VkBool32 (VKAPI_PTR *PFN_vkDebugUtilsMessengerCallbackEXT)(
    VkDebugUtilsMessageSeverityFlagBitsEXT    messageSeverity,
    VkDebugUtilsMessageTypeFlagsEXT           messageTypes,
    const VkDebugUtilsMessengerCallbackDataEXT* pCallbackData,
    void*                                       pUserData);
```

- `messageSeverity` specifies the `VkDebugUtilsMessageSeverityFlagBitsEXT` that triggered this callback.
- `messageTypes` is a bitmask of `VkDebugUtilsMessageTypeFlagBitsEXT` specifying which type of event(s) triggered this callback.
- `pCallbackData` contains all the callback related data in the `VkDebugUtilsMessengerCallbackDataEXT` structure.
- `pUserData` is the user data provided when the `VkDebugUtilsMessengerEXT` was created.

The callback returns a `VkBool32`, which is interpreted in a layer-specified manner. The application **should** always return `VK_FALSE`. The `VK_TRUE` value is reserved for use in layer development.

Valid Usage

- VUID-PFN_vkDebugUtilsMessengerCallbackEXT-None-04769
The callback **must** not make calls to any Vulkan commands

The definition of `VkDebugUtilsMessengerCallbackDataEXT` is:

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsMessengerCallbackDataEXT {
    VkStructureType           sType;
    const void*               pNext;
    VkDebugUtilsMessengerCallbackDataFlagsEXT flags;
    const char*               pMessageIdName;
    int32_t                   messageIdNumber;
    const char*               pMessage;
    uint32_t                  queueLabelCount;
    const VkDebugUtilsLabelEXT* pQueueLabels;
    uint32_t                  cmdBufLabelCount;
    const VkDebugUtilsLabelEXT* pCmdBufLabels;
    uint32_t                  objectCount;
    const VkDebugUtilsObjectNameInfoEXT* pObjects;
} VkDebugUtilsMessengerCallbackDataEXT;
```

- `sType` is a [VkStructureType](#) value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure.
- `flags` is `0` and is reserved for future use.
- `pMessageIdName` is `NULL` or a null-terminated UTF-8 string that identifies the particular message ID that is associated with the provided message. If the message corresponds to a validation layer message, then this string will be the VUID.
- `messageIdNumber` is the ID number of the triggering message. If the message corresponds to a validation layer message, then this number is an internal hash of the VUID.
- `pMessage` is a null-terminated UTF-8 string detailing the trigger conditions. If the message corresponds to a validation layer message, then this will contain the main message with the specification text and link.
- `queueLabelCount` is a count of items contained in the `pQueueLabels` array.
- `pQueueLabels` is `NULL` or a pointer to an array of [VkDebugUtilsLabelEXT](#) active in the current [VkQueue](#) at the time the callback was triggered. Refer to [Queue Labels](#) for more information.
- `cmdBufLabelCount` is a count of items contained in the `pCmdBufLabels` array.
- `pCmdBufLabels` is `NULL` or a pointer to an array of [VkDebugUtilsLabelEXT](#) active in the current [VkCommandBuffer](#) at the time the callback was triggered. Refer to [Command Buffer Labels](#) for more information.
- `objectCount` is a count of items contained in the `pObjects` array.
- `pObjects` is a pointer to an array of [VkDebugUtilsObjectNameInfoEXT](#) objects related to the detected issue. The array is roughly in order of importance, but the 0th element is always guaranteed to be the most important object for this message.

NOTE This structure should only be considered valid during the lifetime of the triggered callback.

Since adding queue and command buffer labels behaves like pushing and popping onto a stack, the

order of both `pQueueLabels` and `pCmdBufLabels` is based on the order the labels were defined. The result is that the first label in either `pQueueLabels` or `pCmdBufLabels` will be the first defined (and therefore the oldest) while the last label in each list will be the most recent.

NOTE

`pQueueLabels` will only be non-NULL if one of the objects in `pObjects` can be related directly to a defined `VkQueue` which has had one or more labels associated with it.

Likewise, `pCmdBufLabels` will only be non-NULL if one of the objects in `pObjects` can be related directly to a defined `VkCommandBuffer` which has had one or more labels associated with it. Additionally, while command buffer labels allow for beginning and ending across different command buffers, the debug messaging framework **cannot** guarantee that labels in `pCmdBufLabels` will contain those defined outside of the associated command buffer. This is partially due to the fact that the association of one command buffer with another may not have been defined at the time the debug message is triggered.

Valid Usage (Implicit)

- VUID-VkDebugUtilsMessengerCallbackDataEXT-sType-sType
`sType` **must** be `VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CALLBACK_DATA_EXT`
- VUID-VkDebugUtilsMessengerCallbackDataEXT-pNext-pNext
`pNext` **must** be NULL
- VUID-VkDebugUtilsMessengerCallbackDataEXT-flags-zerobitmask
`flags` **must** be 0
- VUID-VkDebugUtilsMessengerCallbackDataEXT-pMessageIdName-parameter
If `pMessageIdName` is not NULL, `pMessageIdName` **must** be a null-terminated UTF-8 string
- VUID-VkDebugUtilsMessengerCallbackDataEXT-pMessage-parameter
If `pMessage` is not NULL, `pMessage` **must** be a null-terminated UTF-8 string
- VUID-VkDebugUtilsMessengerCallbackDataEXT-pQueueLabels-parameter
If `queueLabelCount` is not 0, `pQueueLabels` **must** be a valid pointer to an array of `queueLabelCount` valid `VkDebugUtilsLabelEXT` structures
- VUID-VkDebugUtilsMessengerCallbackDataEXT-pCmdBufLabels-parameter
If `cmdBufLabelCount` is not 0, `pCmdBufLabels` **must** be a valid pointer to an array of `cmdBufLabelCount` valid `VkDebugUtilsLabelEXT` structures
- VUID-VkDebugUtilsMessengerCallbackDataEXT-pObjects-parameter
If `objectCount` is not 0, `pObjects` **must** be a valid pointer to an array of `objectCount` valid `VkDebugUtilsObjectNameInfoEXT` structures

```
// Provided by VK_EXT_debug_utils
typedef VkFlags VkDebugUtilsMessengerCallbackDataFlagsEXT;
```

`VkDebugUtilsMessengerCallbackDataFlagsEXT` is a bitmask type for setting a mask, but is currently reserved for future use.

To intentionally submit a debug message, call:

```
// Provided by VK_EXT_debug_utils
void vkSubmitDebugUtilsMessageEXT(
    VkInstance                                instance,
    VkDebugUtilsMessageSeverityFlagBitsEXT    messageSeverity,
    VkDebugUtilsMessageTypeFlagsEXT           messageTypes,
    const VkDebugUtilsMessengerCallbackDataEXT* pCallbackData);
```

- **instance** is the debug stream's **VkInstance**.
- **messageSeverity** is a **VkDebugUtilsMessageSeverityFlagBitsEXT** value specifying the severity of this event/message.
- **messageTypes** is a bitmask of **VkDebugUtilsMessageTypeFlagBitsEXT** specifying which type of event(s) to identify with this message.
- **pCallbackData** contains all the callback related data in the **VkDebugUtilsMessengerCallbackDataEXT** structure.

The call will propagate through the layers and generate callback(s) as indicated by the message's flags. The parameters are passed on to the callback in addition to the **pUserData** value that was defined at the time the messenger was registered.

Valid Usage

- VUID-vkSubmitDebugUtilsMessageEXT-objectType-02591
The **objectType** member of each element of **pCallbackData->pObjects** **must** not be **VK_OBJECT_TYPE_UNKNOWN**

Valid Usage (Implicit)

- VUID-vkSubmitDebugUtilsMessageEXT-instance-parameter
instance **must** be a valid **VkInstance** handle
- VUID-vkSubmitDebugUtilsMessageEXT-messageSeverity-parameter
messageSeverity **must** be a valid **VkDebugUtilsMessageSeverityFlagBitsEXT** value
- VUID-vkSubmitDebugUtilsMessageEXT-messageTypes-parameter
messageTypes **must** be a valid combination of **VkDebugUtilsMessageTypeFlagBitsEXT** values
- VUID-vkSubmitDebugUtilsMessageEXT-messageTypes-requiredbitmask
messageTypes **must** not be **0**
- VUID-vkSubmitDebugUtilsMessageEXT-pCallbackData-parameter
pCallbackData **must** be a valid pointer to a valid **VkDebugUtilsMessengerCallbackDataEXT** structure

To destroy a **VkDebugUtilsMessengerEXT** object, call:

```
// Provided by VK_EXT_debug_utils
void vkDestroyDebugUtilsMessengerEXT(
    VkInstance                                  instance,
    VkDebugUtilsMessengerEXT                   messenger,
    const VkAllocationCallbacks*               pAllocator);
```

- **instance** is the instance where the callback was created.
- **messenger** is the [VkDebugUtilsMessengerEXT](#) object to destroy. **messenger** is an externally synchronized object and **must** not be used on more than one thread at a time. This means that **vkDestroyDebugUtilsMessengerEXT** **must** not be called when a callback is active.
- **pAllocator** controls host memory allocation as described in the [Memory Allocation](#) chapter.

Valid Usage (Implicit)

- VUID-vkDestroyDebugUtilsMessengerEXT-instance-parameter
instance **must** be a valid [VkInstance](#) handle
- VUID-vkDestroyDebugUtilsMessengerEXT-messenger-parameter
If **messenger** is not [VK_NULL_HANDLE](#), **messenger** **must** be a valid [VkDebugUtilsMessengerEXT](#) handle
- VUID-vkDestroyDebugUtilsMessengerEXT-pAllocator-null
pAllocator **must** be [NULL](#)
- VUID-vkDestroyDebugUtilsMessengerEXT-messenger-parent
If **messenger** is a valid handle, it **must** have been created, allocated, or retrieved from **instance**

Host Synchronization

- Host access to **messenger** **must** be externally synchronized

The application **must** ensure that [vkDestroyDebugUtilsMessengerEXT](#) is not executed in parallel with any Vulkan command that is also called with **instance** or child of **instance** as the dispatchable argument.

37.2. Fault Handling

The fault handling mechanism provides a method for the implementation to pass fault information to the application. A fault indicates that an issue has occurred with the host or device that could impact the implementation's ability to function correctly. It consists of a [VkFaultData](#) structure that is used to communicate information about the fault between the implementation and the application, with two methods to obtain the data. The application **can** obtain the fault data from the implementation using [vkGetFaultData](#). Alternatively, the implementation **can** directly call a pre-registered fault handler function ([PFN_vkFaultCallbackFunction](#)) in the application when a fault occurs.

The `VkFaultData` structure provides categories the implementation **must** set to provide basic information on a fault. These allow the implementation to provide a coarse classification of a fault to the application. As the potential faults that could occur will vary between different platforms, it is expected that an implementation would also provide additional implementation-specific data on the fault, enabling the application to take appropriate action.

The implementation **must** also define whether a particular fault results in the fault callback function being called, is communicated via `vkGetFaultData`, or both. This will be decided by several factors including:

- the severity of the fault,
- the application's ability to handle the fault, and
- how the application should handle the fault.

The implementation **must** document the implementation-specific fault data, how the faults are communicated, and expected responses from the application for each of the faults that it **can** report.

37.2.1. Fault Data

The information on a single fault is returned using the `VkFaultData` structure. The `VkFaultData` structure is defined as:

```
// Provided by VKSC_VERSION_1_0
typedef struct VkFaultData {
    VkStructureType    sType;
    void*              pNext;
    VkFaultLevel       faultLevel;
    VkFaultType        faultType;
} VkFaultData;
```

- `sType` is a `VkStructureType` value identifying this structure.
- `pNext` is `NULL` or a pointer to a structure extending this structure that provides implementation-specific data on the fault.
- `faultLevel` is a `VkFaultLevel` that provides the severity of the fault.
- `faultType` is a `VkFaultType` that provides the type of the fault.

To retrieve implementation-specific fault data, `pNext` **can** point to one or more implementation-defined fault structures or `NULL` to not retrieve implementation-specific data.

Valid Usage

- VUID-VkFaultData-pNext-05019
`pNext` **must** be `NULL` or a valid pointer to an implementation-specific structure

Valid Usage (Implicit)

- VUID-VkFaultData-sType-sType
sType **must** be VK_STRUCTURE_TYPE_FAULT_DATA

Possible values of `VkFaultData::faultLevel`, specifying the fault severity, are:

```
// Provided by VKSC_VERSION_1_0
typedef enum VkFaultLevel {
    VK_FAULT_LEVEL_UNASSIGNED = 0,
    VK_FAULT_LEVEL_CRITICAL = 1,
    VK_FAULT_LEVEL_RECOVERABLE = 2,
    VK_FAULT_LEVEL_WARNING = 3,
} VkFaultLevel;
```

- `VK_FAULT_LEVEL_UNASSIGNED` A fault level has not been assigned.
- `VK_FAULT_LEVEL_CRITICAL` A fault that **cannot** be recovered by the application.
- `VK_FAULT_LEVEL_RECOVERABLE` A fault that **can** be recovered by the application.
- `VK_FAULT_LEVEL_WARNING` A fault that indicates a non-optimal condition has occurred, but no recovery is necessary at this point.

Possible values of `VkFaultData::faultType`, specifying the fault type, are:

```
// Provided by VKSC_VERSION_1_0
typedef enum VkFaultType {
    VK_FAULT_TYPE_INVALID = 0,
    VK_FAULT_TYPE_UNASSIGNED = 1,
    VK_FAULT_TYPE_IMPLEMENTATION = 2,
    VK_FAULT_TYPE_SYSTEM = 3,
    VK_FAULT_TYPE_PHYSICAL_DEVICE = 4,
    VK_FAULT_TYPE_COMMAND_BUFFER_FULL = 5,
    VK_FAULT_TYPE_INVALID_API_USAGE = 6,
} VkFaultType;
```

- `VK_FAULT_TYPE_INVALID` The fault data does not contain a valid fault.
- `VK_FAULT_TYPE_UNASSIGNED` A fault type has not been assigned.
- `VK_FAULT_TYPE_IMPLEMENTATION` Implementation-defined fault.
- `VK_FAULT_TYPE_SYSTEM` A fault occurred in the system components.
- `VK_FAULT_TYPE_PHYSICAL_DEVICE` A fault occurred with the physical device.
- `VK_FAULT_TYPE_COMMAND_BUFFER_FULL` Command buffer memory was exhausted before `vkEndCommandBuffer` was called.
- `VK_FAULT_TYPE_INVALID_API_USAGE` Invalid usage of the API was detected by the implementation.

37.2.2. Querying Fault Status

To query the number of current faults and obtain the fault data, call [vkGetFaultData](#).

```
// Provided by VKSC_VERSION_1_0
VkResult vkGetFaultData(
    VkDevice                device,
    VkFaultQueryBehavior    faultQueryBehavior,
    VkBool32*               pUnrecordedFaults,
    uint32_t*               pFaultCount,
    VkFaultData*            pFaults);
```

- **device** is the logical device to obtain faults from.
- **faultQueryBehavior** is a [VkFaultQueryBehavior](#) that specifies the types of faults to obtain from the implementation, and how those faults should be handled.
- **pUnrecordedFaults** is a return boolean that specifies if the logged fault information is incomplete and does not contain entries for all faults that have been detected by the implementation and **may** be reported via [vkGetFaultData](#).
- **pFaultCount** is a pointer to an integer that specifies the number of fault entries.
- **pFaults** is either **NULL** or a pointer to an array of **pFaultCount** [VkFaultData](#) structures to be updated with the recorded fault data.

Access to fault data is internally synchronized, meaning [vkGetFaultData](#) **can** be called from multiple threads simultaneously.

The implementation **must** not record more than **maxQueryFaultCount** faults to be reported by [vkGetFaultData](#).

pUnrecordedFaults is **VK_TRUE** if the implementation has detected one or more faults since the last successful retrieval of fault data using this command, but was unable to record fault information for all faults. Otherwise, **pUnrecordedFaults** is **VK_FALSE**.

If **pFaults** is **NULL**, then the number of faults with the specified **faultQueryBehavior** characteristics associated with **device** is returned in **pFaultCount**, and **pUnrecordedFaults** is set as indicated above. Otherwise, **pFaultCount** **must** point to a variable set by the application to the number of elements in the **pFaults** array, and on return the variable is overwritten with the number of faults actually written to **pFaults**. If **pFaultCount** is less than the number of recorded **device** faults with the specified **faultQueryBehavior** characteristics, at most **pFaultCount** faults will be written, and **VK_INCOMPLETE** will be returned instead of **VK_SUCCESS**, to indicate that not all the available faults were returned.

If **VK_SUCCESS** is returned, the fault information stored by the implementation for the faults that were returned will be handled as specified by **faultQueryBehavior**. Otherwise, the stored fault information is not modified.

For each filled **pFaults** entry, if **pNext** is not **NULL**, the implementation will fill in any implementation-specific structures applicable to that fault that are included in the **pNext** chain.

NOTE

In order to simplify the application logic, an application could have a static allocation sized to `maxQueryFaultCount` which it passes in to each call of `vkGetFaultData`. This allows an application to obtain all the faults available at this time in a single call to `vkGetFaultData`. Furthermore, under this usage pattern, the command will never return `VK_INCOMPLETE`.

If `VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations` is `VK_TRUE`, `vkGetFaultData` **must** not return `VK_ERROR_OUT_OF_HOST_MEMORY`.

Valid Usage

- VUID-vkGetFaultData-pFaultCount-05020
`pFaultCount` **must** be less than or equal to `maxQueryFaultCount`

Valid Usage (Implicit)

- VUID-vkGetFaultData-device-parameter
`device` **must** be a valid `VkDevice` handle
- VUID-vkGetFaultData-faultQueryBehavior-parameter
`faultQueryBehavior` **must** be a valid `VkFaultQueryBehavior` value
- VUID-vkGetFaultData-pUnrecordedFaults-parameter
`pUnrecordedFaults` **must** be a valid pointer to a `VkBool32` value
- VUID-vkGetFaultData-pFaultCount-parameter
`pFaultCount` **must** be a valid pointer to a `uint32_t` value
- VUID-vkGetFaultData-pFaults-parameter
If the value referenced by `pFaultCount` is not 0, and `pFaults` is not `NULL`, `pFaults` **must** be a valid pointer to an array of `pFaultCount` `VkFaultData` structures

Return Codes

Success

- `VK_INCOMPLETE`
- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_UNKNOWN`
- `VK_ERROR_VALIDATION_FAILED`

Possible values that **can** be set in `VkFaultQueryBehavior`, specifying which faults to return, are:

```
// Provided by VKSC_VERSION_1_0
typedef enum VkFaultQueryBehavior {
    VK_FAULT_QUERY_BEHAVIOR_GET_AND_CLEAR_ALL_FAULTS = 0,
} VkFaultQueryBehavior;
```

- **VK_FAULT_QUERY_BEHAVIOR_GET_AND_CLEAR_ALL_FAULTS** All fault types and severities are reported and are cleared from the internal fault storage after retrieval.

37.2.3. Fault Callback

The **VkFaultCallbackInfo** structure allows an application to register a function at device creation that the implementation can call to report faults when they occur. A callback function is registered by attaching a valid **VkFaultCallbackInfo** structure to the **pNext** chain of the **VkDeviceCreateInfo** structure. The callback function is only called by the implementation during a call to the API, using the same thread that is making the API call. The **VkFaultCallbackInfo** structure provides the function pointer to be called by the implementation, and optionally, application memory to store fault data.

The **VkFaultCallbackInfo** structure is defined as:

```
// Provided by VKSC_VERSION_1_0
typedef struct VkFaultCallbackInfo {
    VkStructureType      sType;
    const void*          pNext;
    uint32_t             faultCount;
    VkFaultData*          pFaults;
    PFN_vkFaultCallbackFunction pfnFaultCallback;
} VkFaultCallbackInfo;
```

- **sType** is a **VkStructureType** value identifying this structure.
- **pNext** is **NULL** or pointer to a structure extending this structure.
- **faultCount** is the number of reported faults in the array pointed to by **pFaults**.
- **pFaults** is either **NULL** or a pointer to an array of **faultCount** **VkFaultData** structures.
- **pfnFaultCallback** is a function pointer to the fault handler function that will be called by the implementation when a fault occurs.

If provided, the implementation **may** make use of the **pFaults** array to return fault data to the application when using the fault callback.

NOTE

Prior to Vulkan SC 1.0.11, the application was required to provide the **pFaults** array for fault callback data. This proved to be unwieldy for both applications and implementations and it was made optional as of version 1.0.11. It is expected that most implementations will ignore this and use stack or other preallocated memory for fault callback parameters.

If provided, the application memory referenced by **pFaults** **must** remain accessible throughout the lifetime of the logical device that was created with this structure.

NOTE

The memory pointed to by **pFaults** will be updated by the implementation and should not be used or accessed by the application outside of the fault handling function pointed to by **pfnFaultCallback**. This restriction also applies to any implementation-specific structure chained to an element of **pFaults** by **pNext**.

It is expected that implementations will maintain separate storage for fault information and populate the array pointed to by **pFaults** ahead of calling the fault callback function.

Valid Usage

- VUID-VkFaultCallbackInfo-faultCount-05138
faultCount **must** either be 0, or equal to **VkPhysicalDeviceVulkanSC10Properties::maxCallbackFaultCount**

Valid Usage (Implicit)

- VUID-VkFaultCallbackInfo-sType-sType
sType **must** be **VK_STRUCTURE_TYPE_FAULT_CALLBACK_INFO**
- VUID-VkFaultCallbackInfo-pFaults-parameter
If **faultCount** is not 0, and **pFaults** is not NULL, **pFaults** **must** be a valid pointer to an array of **faultCount** **VkFaultData** structures
- VUID-VkFaultCallbackInfo-pfnFaultCallback-parameter
pfnFaultCallback **must** be a valid **PFN_vkFaultCallbackFunction** value

The function pointer **PFN_vkFaultCallbackFunction** is defined as:

```
// Provided by VKSC_VERSION_1_0
typedef void (VKAPI_PTR *PFN_vkFaultCallbackFunction)(
    VkBool32                unrecordedFaults,
    uint32_t                faultCount,
    const VkFaultData*      pFaults);
```

- **unrecordedFaults** is a boolean that specifies if the supplied fault information is incomplete and does not contain entries for all faults that have been detected by the implementation and **may** be reported via **PFN_vkFaultCallbackFunction** since the last call to this callback.
- **faultCount** will contain the number of reported faults in the array pointed to by **pFaults**.
- **pFaults** will point to an array of **faultCount** **VkFaultData** structures containing the fault information.

An implementation **must** only make calls to **pfnFaultCallback** during the execution of an API

command. An implementation **must** only make calls into the application-provided fault callback from the same thread that called the API command. The implementation **should** not synchronize calls to the callback. If synchronization is needed, the callback **must** provide it.

The fault callback **must** not call any Vulkan commands.

It is implementation-dependent whether faults reported by this callback are also reported via [vkGetFaultData](#), but each unique fault will be reported by at most one callback.

Appendix A: Vulkan Environment for SPIR-V

Shaders for Vulkan are defined by the [Khronos SPIR-V Specification](#) as well as the [Khronos SPIR-V Extended Instructions for GLSL](#) Specification. This appendix defines additional SPIR-V requirements applying to Vulkan shaders.

Versions and Formats

A Vulkan 1.2 implementation **must** support the 1.0, 1.1, 1.2, 1.3, 1.4, and 1.5 versions of SPIR-V and the 1.0 version of the SPIR-V Extended Instructions for GLSL.

A SPIR-V module is interpreted as a series of 32-bit words in host endianness, with literal strings packed as described in section 2.2 of the SPIR-V Specification. The first few words of the SPIR-V module **must** be a magic number and a SPIR-V version number, as described in section 2.3 of the SPIR-V Specification.

Capabilities

The [table below](#) lists the set of SPIR-V capabilities that **may** be supported in Vulkan implementations. The application **must** not select a pipeline cache entry, which was created by passing a SPIR-V module using any of these capabilities to the [offline pipeline cache compiler](#), in a [vkCreate*Pipelines](#) command unless one of the following conditions is met for the [VkDevice](#) specified in the [device](#) parameter of the [vkCreate*Pipelines](#) command:

- The corresponding field in the table is blank.
- Any corresponding Vulkan feature is enabled.
- Any corresponding Vulkan extension is enabled.
- Any corresponding Vulkan property is supported.
- The corresponding core version is supported (as returned by [VkPhysicalDeviceProperties::apiVersion](#)).

Table 79. List of SPIR-V Capabilities and Corresponding Vulkan Features, Extensions, or Core Version

SPIR-V OpCapability	Vulkan feature, extension, or core version
Matrix	VK_VERSION_1_0
Shader	VK_VERSION_1_0
InputAttachment	VK_VERSION_1_0
Sampled1D	VK_VERSION_1_0

SPIR-V OpCapability Vulkan feature, extension, or core version
Image1D VK_VERSION_1_0
SampledBuffer VK_VERSION_1_0
ImageBuffer VK_VERSION_1_0
ImageQuery VK_VERSION_1_0
DerivativeControl VK_VERSION_1_0
Geometry VkPhysicalDeviceFeatures::geometryShader
Tessellation VkPhysicalDeviceFeatures::tessellationShader
Float64 VkPhysicalDeviceFeatures::shaderFloat64
Int64 VkPhysicalDeviceFeatures::shaderInt64
Int64Atomics VkPhysicalDeviceVulkan12Features::shaderBufferInt64Atomics VkPhysicalDeviceVulkan12Features::shaderSharedInt64Atomics VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT::shaderImageInt64Atomics
AtomicFloat32AddEXT VkPhysicalDeviceShaderAtomicFloatFeaturesEXT::shaderBufferFloat32AtomicAdd VkPhysicalDeviceShaderAtomicFloatFeaturesEXT::shaderSharedFloat32AtomicAdd VkPhysicalDeviceShaderAtomicFloatFeaturesEXT::shaderImageFloat32AtomicAdd
AtomicFloat64AddEXT VkPhysicalDeviceShaderAtomicFloatFeaturesEXT::shaderBufferFloat64AtomicAdd VkPhysicalDeviceShaderAtomicFloatFeaturesEXT::shaderSharedFloat64AtomicAdd
Int64ImageEXT VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT::shaderImageInt64Atomics
Int16 VkPhysicalDeviceFeatures::shaderInt16
TessellationPointSize VkPhysicalDeviceFeatures::shaderTessellationAndGeometryPointSize
GeometryPointSize VkPhysicalDeviceFeatures::shaderTessellationAndGeometryPointSize
ImageGatherExtended VkPhysicalDeviceFeatures::shaderImageGatherExtended

SPIR-V OpCapability	Vulkan feature, extension, or core version
StorageImageMultisample	VkPhysicalDeviceFeatures::shaderStorageImageMultisample
UniformBufferArrayDynamicIndexing	VkPhysicalDeviceFeatures::shaderUniformBufferArrayDynamicIndexing
SampledImageArrayDynamicIndexing	VkPhysicalDeviceFeatures::shaderSampledImageArrayDynamicIndexing
StorageBufferArrayDynamicIndexing	VkPhysicalDeviceFeatures::shaderStorageBufferArrayDynamicIndexing
StorageImageArrayDynamicIndexing	VkPhysicalDeviceFeatures::shaderStorageImageArrayDynamicIndexing
ClipDistance	VkPhysicalDeviceFeatures::shaderClipDistance
CullDistance	VkPhysicalDeviceFeatures::shaderCullDistance
ImageCubeArray	VkPhysicalDeviceFeatures::imageCubeArray
SampleRateShading	VkPhysicalDeviceFeatures::sampleRateShading
SparseResidency	VkPhysicalDeviceFeatures::shaderResourceResidency
MinLod	VkPhysicalDeviceFeatures::shaderResourceMinLod
SampledCubeArray	VkPhysicalDeviceFeatures::imageCubeArray
ImageMSArray	VkPhysicalDeviceFeatures::shaderStorageImageMultisample
StorageImageExtendedFormats	VK_VERSION_1_0
InterpolationFunction	VkPhysicalDeviceFeatures::sampleRateShading
StorageImageReadWithoutFormat	VkPhysicalDeviceFeatures::shaderStorageImageReadWithoutFormat
StorageImageWriteWithoutFormat	VkPhysicalDeviceFeatures::shaderStorageImageWriteWithoutFormat
MultiViewport	VkPhysicalDeviceFeatures::multiViewport

SPIR-V OpCapability Vulkan feature, extension, or core version
DrawParameters VkPhysicalDeviceVulkan11Features::shaderDrawParameters VkPhysicalDeviceShaderDrawParametersFeatures::shaderDrawParameters
MultiView VkPhysicalDeviceVulkan11Features::multiview
DeviceGroup VK_VERSION_1_1
VariablePointersStorageBuffer VkPhysicalDeviceVulkan11Features::variablePointersStorageBuffer
VariablePointers VkPhysicalDeviceVulkan11Features::variablePointers
ShaderClockKHR VK_KHR_shader_clock
StencilExportEXT VK_EXT_shader_stencil_export
ShaderViewportIndex VkPhysicalDeviceVulkan12Features::shaderOutputViewportIndex
ShaderLayer VkPhysicalDeviceVulkan12Features::shaderOutputLayer
StorageBuffer16BitAccess VkPhysicalDeviceVulkan11Features::storageBuffer16BitAccess
UniformAndStorageBuffer16BitAccess VkPhysicalDeviceVulkan11Features::uniformAndStorageBuffer16BitAccess
StoragePushConstant16 VkPhysicalDeviceVulkan11Features::storagePushConstant16
StorageInputOutput16 VkPhysicalDeviceVulkan11Features::storageInputOutput16
GroupNonUniform VK_SUBGROUP_FEATURE_BASIC_BIT
GroupNonUniformVote VK_SUBGROUP_FEATURE_VOTE_BIT
GroupNonUniformArithmetic VK_SUBGROUP_FEATURE_ARITHMETIC_BIT
GroupNonUniformBallot VK_SUBGROUP_FEATURE_BALLOT_BIT
GroupNonUniformShuffle VK_SUBGROUP_FEATURE_SHUFFLE_BIT

SPIR-V OpCapability	Vulkan feature, extension, or core version
GroupNonUniformShuffleRelative	VK_SUBGROUP_FEATURE_SHUFFLE_RELATIVE_BIT
GroupNonUniformClustered	VK_SUBGROUP_FEATURE_CLUSTERED_BIT
GroupNonUniformQuad	VK_SUBGROUP_FEATURE_QUAD_BIT
SampleMaskPostDepthCoverage	VK_EXT_post_depth_coverage
ShaderNonUniform	VK_VERSION_1_2
RuntimeDescriptorArray	VkPhysicalDeviceVulkan12Features::runtimeDescriptorArray
InputAttachmentArrayDynamicIndexing	VkPhysicalDeviceVulkan12Features::shaderInputAttachmentArrayDynamicIndexing
UniformTexelBufferArrayDynamicIndexing	VkPhysicalDeviceVulkan12Features::shaderUniformTexelBufferArrayDynamicIndexing
StorageTexelBufferArrayDynamicIndexing	VkPhysicalDeviceVulkan12Features::shaderStorageTexelBufferArrayDynamicIndexing
UniformBufferArrayNonUniformIndexing	VkPhysicalDeviceVulkan12Features::shaderUniformBufferArrayNonUniformIndexing
SampledImageArrayNonUniformIndexing	VkPhysicalDeviceVulkan12Features::shaderSampledImageArrayNonUniformIndexing
StorageBufferArrayNonUniformIndexing	VkPhysicalDeviceVulkan12Features::shaderStorageBufferArrayNonUniformIndexing
StorageImageArrayNonUniformIndexing	VkPhysicalDeviceVulkan12Features::shaderStorageImageArrayNonUniformIndexing
InputAttachmentArrayNonUniformIndexing	VkPhysicalDeviceVulkan12Features::shaderInputAttachmentArrayNonUniformIndexing
UniformTexelBufferArrayNonUniformIndexing	VkPhysicalDeviceVulkan12Features::shaderUniformTexelBufferArrayNonUniformIndexing
StorageTexelBufferArrayNonUniformIndexing	VkPhysicalDeviceVulkan12Features::shaderStorageTexelBufferArrayNonUniformIndexing
FragmentFullyCoveredEXT	VK_EXT_conservative_rasterization
Float16	VkPhysicalDeviceVulkan12Features::shaderFloat16
Int8	VkPhysicalDeviceVulkan12Features::shaderInt8

SPIR-V OpCapability Vulkan feature, extension, or core version
StorageBuffer8BitAccess VkPhysicalDeviceVulkan12Features::storageBuffer8BitAccess
UniformAndStorageBuffer8BitAccess VkPhysicalDeviceVulkan12Features::uniformAndStorageBuffer8BitAccess
StoragePushConstant8 VkPhysicalDeviceVulkan12Features::storagePushConstant8
VulkanMemoryModel VkPhysicalDeviceVulkan12Features::vulkanMemoryModel
VulkanMemoryModelDeviceScope VkPhysicalDeviceVulkan12Features::vulkanMemoryModelDeviceScope
DenormPreserve VkPhysicalDeviceVulkan12Properties::shaderDenormPreserveFloat16 VkPhysicalDeviceVulkan12Properties::shaderDenormPreserveFloat32 VkPhysicalDeviceVulkan12Properties::shaderDenormPreserveFloat64
DenormFlushToZero VkPhysicalDeviceVulkan12Properties::shaderDenormFlushToZeroFloat16 VkPhysicalDeviceVulkan12Properties::shaderDenormFlushToZeroFloat32 VkPhysicalDeviceVulkan12Properties::shaderDenormFlushToZeroFloat64
SignedZeroInfNanPreserve VkPhysicalDeviceVulkan12Properties::shaderSignedZeroInfNanPreserveFloat16 VkPhysicalDeviceVulkan12Properties::shaderSignedZeroInfNanPreserveFloat32 VkPhysicalDeviceVulkan12Properties::shaderSignedZeroInfNanPreserveFloat64
RoundingModeRTE VkPhysicalDeviceVulkan12Properties::shaderRoundingModeRTEFloat16 VkPhysicalDeviceVulkan12Properties::shaderRoundingModeRTEFloat32 VkPhysicalDeviceVulkan12Properties::shaderRoundingModeRTEFloat64
RoundingModeRTZ VkPhysicalDeviceVulkan12Properties::shaderRoundingModeRTZFloat16 VkPhysicalDeviceVulkan12Properties::shaderRoundingModeRTZFloat32 VkPhysicalDeviceVulkan12Properties::shaderRoundingModeRTZFloat64
PhysicalStorageBufferAddresses VkPhysicalDeviceVulkan12Features::bufferDeviceAddress
FragmentShaderSampleInterlockEXT VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT::fragmentShaderSampleInterlock
FragmentShaderPixelInterlockEXT VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT::fragmentShaderPixelInterlock
FragmentShaderShadingRateInterlockEXT VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT::fragmentShaderShadingRateInterlock

SPIR-V OpCapability
Vulkan feature, extension, or core version
DemoteToHelperInvocation
VkPhysicalDeviceVulkan13Features::shaderDemoteToHelperInvocation
VkPhysicalDeviceShaderDemoteToHelperInvocationFeaturesEXT
::shaderDemoteToHelperInvocation
FragmentShadingRateKHR
VkPhysicalDeviceFragmentShadingRateFeaturesKHR::pipelineFragmentShadingRate
VkPhysicalDeviceFragmentShadingRateFeaturesKHR::primitiveFragmentShadingRate
VkPhysicalDeviceFragmentShadingRateFeaturesKHR::attachmentFragmentShadingRate

The application **must** not select a pipeline cache entry, which was created by passing a SPIR-V module containing any of the following to the [offline pipeline cache compiler](#), containing any of the following in a `vkCreate*Pipelines` command:

- any `OpCapability` not listed above,
- an unsupported capability, or
- a capability which corresponds to a Vulkan feature or extension which has not been enabled.

SPIR-V Extensions

The [following table](#) lists SPIR-V extensions that implementations **may** support. The application **must** not select a pipeline cache entry, which was created by passing a SPIR-V module using any of the following SPIR-V extensions to the [offline pipeline cache compiler](#), in a `vkCreate*Pipelines` command unless one of the following conditions is met for the `VkDevice` specified in the `device` parameter of the `vkCreate*Pipelines` command:

- Any corresponding Vulkan extension is enabled.
- The corresponding core version is supported (as returned by `VkPhysicalDeviceProperties::apiVersion`).

Table 80. List of SPIR-V Extensions and Corresponding Vulkan Extensions or Core Version

SPIR-V OpExtension
Vulkan extension or core version
SPV_KHR_variable_pointers
VK_VERSION_1_1
SPV_KHR_shader_draw_parameters
VK_VERSION_1_1
SPV_KHR_8bit_storage
VK_VERSION_1_2
SPV_KHR_16bit_storage
VK_VERSION_1_1
SPV_KHR_shader_clock
VK_KHR_shader_clock

SPIR-V OpExtension Vulkan extension or core version
SPV_KHR_float_controls VK_VERSION_1_2
SPV_KHR_storage_buffer_storage_class VK_VERSION_1_1
SPV_KHR_post_depth_coverage VK_EXT_post_depth_coverage
SPV_EXT_shader_stencil_export VK_EXT_shader_stencil_export
SPV_EXT_shader_viewport_index_layer VK_VERSION_1_2
SPV_EXT_descriptor_indexing VK_VERSION_1_2
SPV_KHR_vulkan_memory_model VK_VERSION_1_2
SPV_KHR_physical_storage_buffer VK_VERSION_1_2
SPV_EXT_fragment_shader_interlock VK_EXT_fragment_shader_interlock
SPV_EXT_demote_to_helper_invocation VK_EXT_shader_demote_to_helper_invocation
SPV_KHR_fragment_shading_rate VK_KHR_fragment_shading_rate
SPV_EXT_shader_image_int64 VK_EXT_shader_image_atomic_int64
SPV_KHR_terminate_invocation VK_KHR_shader_terminate_invocation
SPV_KHR_multiview VK_VERSION_1_1
SPV_EXT_shader_atomic_float_add VK_EXT_shader_atomic_float
SPV_EXT_fragment_fully_covered VK_EXT_conservative_rasterization
SPV_KHR_device_group VK_VERSION_1_1

Validation Rules Within a Module

Pipeline cache entries **must** have been compiled with the [offline pipeline cache compiler](#) using

SPIR-V modules that conform to the following rules:

Standalone SPIR-V Validation

The following rules **can** be validated with only the SPIR-V module itself. They do not depend on knowledge of the implementation and its capabilities or knowledge of runtime information, such as enabled features.

Valid Usage

- VUID-StandaloneSpirv-None-04633
Every entry point **must** have no return value and accept no arguments
- VUID-StandaloneSpirv-None-04634
The static function-call graph for an entry point **must** not contain cycles; that is, static recursion is not allowed
- VUID-StandaloneSpirv-None-04635
The **Logical** or **PhysicalStorageBuffer64** addressing model **must** be selected
- VUID-StandaloneSpirv-None-04636
Scope for execution **must** be limited to **Workgroup** or **Subgroup**
- VUID-StandaloneSpirv-None-04637
If the **Scope** for execution is **Workgroup**, then it **must** only be used in the task, mesh, tessellation control, or compute **Execution Model**
- VUID-StandaloneSpirv-None-04638
Scope for memory **must** be limited to **Device**, **QueueFamily**, **Workgroup**, **ShaderCallKHR**, **Subgroup**, or **Invocation**
- VUID-StandaloneSpirv-ExecutionModel-07320
If the **Execution Model** is **TessellationControl**, and the **MemoryModel** is **GLSL450**, the **Scope** for memory **must** not be **Workgroup**
- VUID-StandaloneSpirv-None-07321
If the **Scope** for memory is **Workgroup**, then it **must** only be used in the task, mesh, tessellation control, or compute **Execution Model**
- VUID-StandaloneSpirv-None-04640
If the **Scope** for memory is **ShaderCallKHR**, then it **must** only be used in ray generation, intersection, closest hit, any-hit, miss, and callable **Execution Model**
- VUID-StandaloneSpirv-None-04641
If the **Scope** for memory is **Invocation**, then **MemorySemantics** **must** use **Relaxed** memory order
- VUID-StandaloneSpirv-None-04642
Scope for **group operations** **must** be limited to **Subgroup**
- VUID-StandaloneSpirv-SubgroupVoteKHR-07951
If none of the **SubgroupVoteKHR**, **GroupNonUniform**, or **SubgroupBallotKHR** capabilities are declared, **Scope** for memory **must** not be **Subgroup**
- VUID-StandaloneSpirv-None-04643

Storage Class **must** be limited to UniformConstant, Input, Uniform, Output, Workgroup, Private, Function, PushConstant, Image, StorageBuffer, RayPayloadKHR, IncomingRayPayloadKHR, HitAttributeKHR, CallableDataKHR, IncomingCallableDataKHR, ShaderRecordBufferKHR, PhysicalStorageBuffer, or TileImageEXT

- VUID-StandaloneSpirv-None-04644

If the Storage Class is Output, then it **must** not be used in the GLCompute, RayGenerationKHR, IntersectionKHR, AnyHitKHR, ClosestHitKHR, MissKHR, or CallableKHR Execution Model

- VUID-StandaloneSpirv-None-04645

If the Storage Class is Workgroup, then it **must** only be used in the task, mesh, or compute Execution Model

- VUID-StandaloneSpirv-None-08720

If the Storage Class is TileImageEXT, then it **must** only be used in the fragment execution model

- VUID-StandaloneSpirv-MemorySemantics-10865

MemorySemantics **must** have at most one non-relaxed memory order bit set (Acquire, Release, or AcquireRelease)

- VUID-StandaloneSpirv-MemorySemantics-10866

MemorySemantics with SequentiallyConsistent memory order **must** not be used in the Vulkan API

- VUID-StandaloneSpirv-MemorySemantics-10867

MemorySemantics **must** not use Acquire or AcquireRelease memory order with OpAtomicStore

- VUID-StandaloneSpirv-MemorySemantics-10868

MemorySemantics **must** not use Release or AcquireRelease memory order with OpAtomicLoad

- VUID-StandaloneSpirv-MemorySemantics-10869

MemorySemantics **must** not use Relaxed memory order with OpMemoryBarrier

- VUID-StandaloneSpirv-MemorySemantics-10870

MemorySemantics with a non-relaxed memory order (Acquire, Release, or AcquireRelease) **must** have at least one Vulkan-supported storage class semantics bit set (UniformMemory, WorkgroupMemory, ImageMemory, or OutputMemory)

- VUID-StandaloneSpirv-MemorySemantics-10871

MemorySemantics with at least one Vulkan-supported storage class semantics bit set (UniformMemory, WorkgroupMemory, ImageMemory, or OutputMemory) **must** use a non-relaxed memory order (Acquire, Release, or AcquireRelease)

- VUID-StandaloneSpirv-MemorySemantics-10872

MemorySemantics with MakeAvailable bit set **must** use Release or AcquireRelease memory order

- VUID-StandaloneSpirv-MemorySemantics-10873

MemorySemantics with MakeVisible bit set **must** use Acquire or AcquireRelease memory order

- VUID-StandaloneSpirv-MemorySemantics-10874

MemorySemantics with Volatile bit set **must** not be used with barrier instructions (OpControlBarrier or OpMemoryBarrier)

- VUID-StandaloneSpirv-UnequalMemorySemantics-10875
UnequalMemorySemantics of OpAtomicCompareExchange **must** not use Release or AcquireRelease memory order
- VUID-StandaloneSpirv-UnequalMemorySemantics-10876
UnequalMemorySemantics of OpAtomicCompareExchange **must** not use a stronger memory order than the corresponding EqualMemorySemantics
- VUID-StandaloneSpirv-UnequalMemorySemantics-10877
UnequalMemorySemantics of OpAtomicCompareExchange **must** not have any Vulkan-supported storage class semantics bit set (UniformMemory, WorkgroupMemory, ImageMemory, or OutputMemory) unless this bit is also set in the corresponding EqualMemorySemantics
- VUID-StandaloneSpirv-UnequalMemorySemantics-10878
UnequalMemorySemantics of OpAtomicCompareExchange **must** not have MakeVisible bit set unless this bit is also set in the corresponding EqualMemorySemantics
- VUID-StandaloneSpirv-UnequalMemorySemantics-10879
UnequalMemorySemantics of OpAtomicCompareExchange **must** have Volatile bit set if and only if this bit is also set in the corresponding EqualMemorySemantics
- VUID-StandaloneSpirv-OpVariable-04651
Any variable with an Initializer operand **must** have Output, Private, Function, or Workgroup as its Storage Class operand
- VUID-StandaloneSpirv-OpVariable-04734
Any variable with an Initializer operand and Workgroup as its Storage Class operand **must** use OpConstantNull as the initializer
- VUID-StandaloneSpirv-OpReadClockKHR-04652
Scope for OpReadClockKHR **must** be limited to Subgroup or Device
- VUID-StandaloneSpirv-OriginLowerLeft-04653
The OriginLowerLeft Execution Mode **must** not be used; fragment entry points **must** declare OriginUpperLeft
- VUID-StandaloneSpirv-PixelCenterInteger-04654
The PixelCenterInteger Execution Mode **must** not be used (pixels are always centered at half-integer coordinates)
- VUID-StandaloneSpirv-UniformConstant-04655
Any variable in the UniformConstant Storage Class **must** be typed as either OpTypeImage, OpTypeSampler, OpTypeSampledImage, OpTypeAccelerationStructureKHR, OpTypeTensorARM or an array of one of these types
- VUID-StandaloneSpirv-Uniform-06807
Any variable in the Uniform or StorageBuffer Storage Class **must** be typed as OpTypeStruct or an array of this type
- VUID-StandaloneSpirv-PushConstant-06808
Any variable in the PushConstant Storage Class **must** be typed as OpTypeStruct
- VUID-StandaloneSpirv-OpTypeImage-04656
OpTypeImage **must** declare a scalar 32-bit float, 64-bit integer, or 32-bit integer type for the “Sampled Type” (RelaxedPrecision can be applied to a sampling instruction and to the variable holding the result of a sampling instruction)

- VUID-StandaloneSpirv-OpTypeImage-04657
OpTypeImage **must** have a “Sampled” operand of 1 (sampled image) or 2 (storage image)
- VUID-StandaloneSpirv-OpTypeSampledImage-06671
OpTypeSampledImage **must** have a **OpTypeImage** with a “Sampled” operand of 1 (sampled image)
- VUID-StandaloneSpirv-Image-04965
The **SPIR-V Type** of the **Image Format** operand of an **OpTypeImage** **must** match the **Sampled Type**, as defined in **Image Format and Type Matching**
- VUID-StandaloneSpirv-OpImageTexelPointer-04658
If an **OpImageTexelPointer** is used in an atomic operation, the image type of the **image** parameter to **OpImageTexelPointer** **must** have an image format of **R64i**, **R64ui**, **R32f**, **R32i**, or **R32ui**
- VUID-StandaloneSpirv-OpImageQuerySizeLod-04659
OpImageQuerySizeLod, **OpImageQueryLod**, and **OpImageQueryLevels** **must** only consume an “Image” operand whose type has its “Sampled” operand set to 1
- VUID-StandaloneSpirv-OpTypeImage-09638
An **OpTypeImage** **must** not have a “Dim” operand of **Rect**
- VUID-StandaloneSpirv-OpTypeImage-06214
An **OpTypeImage** with a “Dim” operand of **SubpassData** **must** have an “Arrayed” operand of 0 (non-arrayed) and a “Sampled” operand of 2 (storage image)
- VUID-StandaloneSpirv-SubpassData-04660
The (u,v) coordinates used for a **SubpassData** **must** be the <id> of a constant vector (0,0)
- VUID-StandaloneSpirv-OpTypeImage-06924
Objects of types **OpTypeImage**, **OpTypeSampler**, **OpTypeSampledImage**, **OpTypeAccelerationStructureKHR**, **OpTypeTensorARM**, and arrays of these types **must** not be stored to or modified
- VUID-StandaloneSpirv-Uniform-06925
Any variable in the **Uniform Storage Class** decorated as **Block** **must** not be stored to or modified
- VUID-StandaloneSpirv-Offset-04865
Any image instruction which uses an **Offset**, **ConstOffset**, or **ConstOffsets** image operand, **must** only consume a “Sampled Image” operand whose type has its “Sampled” operand set to 1
- VUID-StandaloneSpirv-OpImageGather-04664
The “Component” operand of **OpImageGather**, and **OpImageSparseGather** **must** be the <id> of a constant instruction
- VUID-StandaloneSpirv-OpImage-04777
OpImage*Dref* instructions **must** not consume an image whose **Dim** is 3D
- VUID-StandaloneSpirv-None-04667
Structure types **must** not contain opaque types
- VUID-StandaloneSpirv-BuiltIn-04668
Any **BuiltIn** decoration not listed in **Built-In Variables** **must** not be used

- VUID-StandaloneSpirv-OpEntryPoint-09658
For a given **OpEntryPoint**, any **BuiltIn** decoration **must** not be used more than once by the **Input** interface
- VUID-StandaloneSpirv-OpEntryPoint-09659
For a given **OpEntryPoint**, any **BuiltIn** decoration **must** not be used more than once by the **Output** interface
- VUID-StandaloneSpirv-Location-06672
The **Location** or **Component** decorations **must** only be used with the **Input**, **Output**, **RayPayloadKHR**, **IncomingRayPayloadKHR**, **HitAttributeKHR**, **HitObjectAttributeNV**, **CallableDataKHR**, **IncomingCallableDataKHR**, or **ShaderRecordBufferKHR** storage classes
- VUID-StandaloneSpirv-Location-04915
The **Location** or **Component** decorations **must** not be used with **BuiltIn**
- VUID-StandaloneSpirv-Location-04916
The **Location** decorations **must** be used on **user-defined variables**
- VUID-StandaloneSpirv-Location-04917
If a **user-defined variable** is not a pointer to a **Block** decorated **OpTypeStruct**, then the variable **must** have a **Location** decoration
- VUID-StandaloneSpirv-Location-04918
If a **user-defined variable** has a **Location** decoration, and the variable is a pointer to a **OpTypeStruct**, then the members of that structure **must** not have **Location** decorations
- VUID-StandaloneSpirv-Location-04919
If a **user-defined variable** does not have a **Location** decoration, and the variable is a pointer to a **Block** decorated **OpTypeStruct**, then each member of the struct **must** have a **Location** decoration
- VUID-StandaloneSpirv-Component-04920
The **Component** decoration value **must** not be greater than 3
- VUID-StandaloneSpirv-Component-04921
If the **Component** decoration is used on a variable that has a **OpTypeVector** type with a **Component Type** with a **Width** that is less than or equal to 32, the sum of its **Component Count** and the **Component** decoration value **must** be less than or equal to 4
- VUID-StandaloneSpirv-Component-04922
If the **Component** decoration is used on a variable that has a **OpTypeVector** type with a **Component Type** with a **Width** that is equal to 64, the sum of two times its **Component Count** and the **Component** decoration value **must** be less than or equal to 4
- VUID-StandaloneSpirv-Component-04923
The **Component** decorations value **must** not be 1 or 3 for scalar or two-component 64-bit data types
- VUID-StandaloneSpirv-Component-07703
The **Component** decorations **must** not be used for a 64-bit vector type with more than two components
- VUID-StandaloneSpirv-Component-10583
The **Component** decorations **must** not be used with any type that is not a scalar, vector, array of scalars or vectors, or an array of arrays of scalars or vectors

- VUID-StandaloneSpirv-Execution-10584
If the **Execution Model** of an entry point is not **TessellationControl** or **Geometry**, its interface definition **must** not include any variables in the **Input** storage class decorated with **Component** that are arrays of arrays of scalars or vectors
- VUID-StandaloneSpirv-Execution-10585
If the **Execution Model** of an entry point is not **MeshNV**, its interface definition **must** not include any variables in the **Output** storage class decorated with **Component** that are arrays of arrays of scalars or vectors
- VUID-StandaloneSpirv-Output-10586
Variables in the **Output** storage class in the **Vertex**, **TessellationControl**, **TessellationEvaluation**, or **Geometry** execution model **must** not have overlapping **Component** and **Location** decorations as defined by [Location and Component Assignment](#)
- VUID-StandaloneSpirv-Output-10587
Variables in the **Output** storage class in the **Fragment** execution model **must** not have both identical **Index** decorations and overlapping **Component** and **Location** decorations as defined by [Location and Component Assignment](#)
- VUID-StandaloneSpirv-Input-10588
Variables in the **Input** storage class in the **TessellationControl**, **TessellationEvaluation**, **Geometry**, or **Fragment** execution model **must** not have overlapping **Component** and **Location** decorations as defined by [Location and Component Assignment](#)
- VUID-StandaloneSpirv-Input-10604
Any variable in the **Input** storage class in the **Fragment** execution model **must** not have an assigned **Location** shared with another variable with different decorations for **PerVertexKHR**, **Flat**, **NoPerspective**, **Sample**, or **Centroid**
- VUID-StandaloneSpirv-Input-09557
The pointers of any **Input** or **Output** [Interface user-defined variables](#) **must** not contain any **PhysicalStorageBuffer Storage Class** pointers
- VUID-StandaloneSpirv-None-10684
All variables **must** have valid explicit layout decorations [as described in Shader Interfaces](#)
- VUID-StandaloneSpirv-GLSLShared-04669
The **GLSLShared** and **GLSLPacked** decorations **must** not be used
- VUID-StandaloneSpirv-TessLevelInner-10880
Any variable decorated with **TessLevelInner** or **TessLevelOuter** **must** also be decorated with **Patch**
- VUID-StandaloneSpirv-Flat-04670
The **Flat**, **NoPerspective**, **Sample**, and **Centroid** decorations **must** only be used on variables with the **Output** or **Input Storage Class**
- VUID-StandaloneSpirv-Flat-06201
The **Flat**, **NoPerspective**, **Sample**, and **Centroid** decorations **must** not be used on variables with the **Output** storage class in a fragment shader
- VUID-StandaloneSpirv-Flat-06202
The **Flat**, **NoPerspective**, **Sample**, and **Centroid** decorations **must** not be used on variables with the **Input** storage class in a vertex shader

- VUID-StandaloneSpirv-PerVertexKHR-06777
The **PerVertexKHR** decoration **must** only be used on variables with the **Input Storage Class** in a fragment shader
- VUID-StandaloneSpirv-Flat-04744
Any variable with integer or double-precision floating-point type and with **Input Storage Class** in a fragment shader, **must** be decorated **Flat**
- VUID-StandaloneSpirv-ViewportRelativeNV-04672
The **ViewportRelativeNV** decoration **must** only be used on a variable decorated with **Layer** in the vertex, tessellation evaluation, or geometry shader stages
- VUID-StandaloneSpirv-ViewportRelativeNV-04673
The **ViewportRelativeNV** decoration **must** not be used unless a variable decorated with one of **ViewportIndex** or **ViewportMaskNV** is also statically used by the same **OpEntryPoint**
- VUID-StandaloneSpirv-ViewportMaskNV-04674
The **ViewportMaskNV** and **ViewportIndex** decorations **must** not both be statically used by one or more **OpEntryPoint**'s that form the **pre-rasterization shader stages** of a graphics pipeline
- VUID-StandaloneSpirv-FPRoundingMode-04675
Rounding modes other than round-to-nearest-even and round-towards-zero **must** not be used for the **FPRoundingMode** decoration
- VUID-StandaloneSpirv-Invariant-04677
Variables decorated with **Invariant** and variables with structure types that have any members decorated with **Invariant** **must** be in the **Output** or **Input Storage Class**, **Invariant** used on an **Input Storage Class** variable or structure member has no effect
- VUID-StandaloneSpirv-VulkanMemoryModel-04678
If the **VulkanMemoryModel** capability is not declared, the **Volatile** decoration **must** be used on any variable declaration that includes one of the **SMIDNV**, **WarpIDNV**, **SubgroupSize**, **SubgroupLocalInvocationId**, **SubgroupEqMask**, **SubgroupGeMask**, **SubgroupGtMask**, **SubgroupLeMask**, or **SubgroupLtMask BuiltIn** decorations when used in the ray generation, closest hit, miss, intersection, or callable shaders, or with the **RayTmaxKHR BuiltIn** decoration when used in an intersection shader
- VUID-StandaloneSpirv-VulkanMemoryModel-04679
If the **VulkanMemoryModel** capability is declared, the **OpLoad** instruction **must** use **MemorySemantics** with the **Volatile** flag when it accesses into any variable that includes one of the **SMIDNV**, **WarpIDNV**, **SubgroupSize**, **SubgroupLocalInvocationId**, **SubgroupEqMask**, **SubgroupGeMask**, **SubgroupGtMask**, **SubgroupLeMask**, or **SubgroupLtMask BuiltIn** decorations when used in the ray generation, closest hit, miss, intersection, or callable shaders, or with the **RayTmaxKHR BuiltIn** decoration when used in an intersection shader
- VUID-StandaloneSpirv-OpTypeRuntimeArray-04680
OpTypeRuntimeArray **must** only be instantiated by a variable as:
 - the last member of a **Block**-decorated **OpTypeStruct** in **StorageBuffer** or **PhysicalStorageBuffer** storage **Storage Class**
 - **BufferBlock**-decorated **OpTypeStruct** in the **Uniform** storage **Storage Class**
 - the outermost dimension of an arrayed variable in the **StorageBuffer**, **Uniform**, or **UniformConstant** storage **Storage Class**

- the `NodePayloadAMD` storage `Storage Class` when the `CoalescingAMD` Execution Mode is specified
- VUID-StandaloneSpirv-OpArrayLength-11805
`OpArrayLength` **must** not be used with an `OpTypeRuntimeArray` that is the last member of a `Block`-decorated `OpTypeStruct` in the `Uniform` storage `Storage Class`
- VUID-StandaloneSpirv-Function-04681
A type *T* that is an array sized with a specialization constant **must** neither be, nor be contained in, the type *T2* of a variable *V*, unless either: a) *T* is equal to *T2*, b) *V* is declared in the `Function`, or `Private Storage Class`, c) *V* is a non-Block variable in the `Workgroup Storage Class`, or d) *V* is an interface variable with an additional level of arrayness, as described in interface matching, and *T* is the member type of the array type *T2*
- VUID-StandaloneSpirv-OpControlBarrier-04682
If `OpControlBarrier` is used in ray generation, intersection, any-hit, closest hit, miss, fragment, vertex, tessellation evaluation, or geometry shaders, the execution Scope **must** be `Subgroup`
- VUID-StandaloneSpirv-None-10685
For each compute shader entry point, either a `TileShadingRateQCOM`, `LocalSize`, or `LocalSizeId Execution Mode`, or an object decorated with the `WorkgroupSize` decoration **must** be specified
- VUID-StandaloneSpirv-DerivativeGroupQuadsNV-04684
For compute shaders using the `DerivativeGroupQuadsNV` execution mode, the first two dimensions of the local workgroup size **must** be a multiple of two
- VUID-StandaloneSpirv-DerivativeGroupLinearNV-04778
For compute shaders using the `DerivativeGroupLinearNV` execution mode, the product of the dimensions of the local workgroup size **must** be a multiple of four
- VUID-StandaloneSpirv-DerivativeGroupQuadsKHR-10151
For compute, mesh, or task shaders using the `DerivativeGroupQuadsKHR` execution mode, the first two dimensions of the local workgroup size **must** be a multiple of two
- VUID-StandaloneSpirv-DerivativeGroupLinearKHR-10152
For compute, mesh, or task shaders using the `DerivativeGroupLinearKHR` execution mode, the product of the dimensions of the local workgroup size **must** be a multiple of four
- VUID-StandaloneSpirv-OpGroupNonUniformBallotBitCount-04685
If `OpGroupNonUniformBallotBitCount` is used, the group operation **must** be limited to `Reduce`, `InclusiveScan`, or `ExclusiveScan`
- VUID-StandaloneSpirv-None-04686
The *Pointer* operand of all atomic instructions **must** have a `Storage Class` limited to `Uniform`, `Workgroup`, `Image`, `StorageBuffer`, `PhysicalStorageBuffer`, or `TaskPayloadWorkgroupEXT`
- VUID-StandaloneSpirv-Offset-04687
Output variables or block members decorated with `Offset` that have a 64-bit type, or a composite type containing a 64-bit type, **must** specify an `Offset` value aligned to a 8 byte boundary
- VUID-StandaloneSpirv-Offset-04689
The size of any output block containing any member decorated with `Offset` that is a 64-bit

type **must** be a multiple of 8

- VUID-StandaloneSpirv-Offset-04690

The first member of an output block specifying a **Offset** decoration **must** specify a **Offset** value that is aligned to an 8 byte boundary if that block contains any member decorated with **Offset** and is a 64-bit type

- VUID-StandaloneSpirv-Offset-04691

Output variables or block members decorated with **Offset** that have a 32-bit type, or a composite type contains a 32-bit type, **must** specify an **Offset** value aligned to a 4 byte boundary

- VUID-StandaloneSpirv-Offset-04692

Output variables, blocks or block members decorated with **Offset** **must** only contain base types that have components that are either 32-bit or 64-bit in size

- VUID-StandaloneSpirv-Offset-04716

Only variables or block members in the output interface decorated with **Offset** **can** be captured for transform feedback, and those variables or block members **must** also be decorated with **XfbBuffer** and **XfbStride**, or inherit **XfbBuffer** and **XfbStride** decorations from a block containing them

- VUID-StandaloneSpirv-XfbBuffer-04693

All variables or block members in the output interface of the entry point being compiled decorated with a specific **XfbBuffer** value **must** all be decorated with identical **XfbStride** values

- VUID-StandaloneSpirv-Stream-04694

If any variables or block members in the output interface of the entry point being compiled are decorated with **Stream**, then all variables belonging to the same **XfbBuffer** **must** specify the same **Stream** value

- VUID-StandaloneSpirv-XfbBuffer-04696

For any two variables or block members in the output interface of the entry point being compiled with the same **XfbBuffer** value, the ranges determined by the **Offset** decoration and the size of the type **must** not overlap

- VUID-StandaloneSpirv-XfbBuffer-04697

All block members in the output interface of the entry point being compiled that are in the same block and have a declared or inherited **XfbBuffer** decoration **must** specify the same **XfbBuffer** value

- VUID-StandaloneSpirv-RayPayloadKHR-04698

RayPayloadKHR Storage Class **must** only be used in ray generation, closest hit or miss shaders

- VUID-StandaloneSpirv-IncomingRayPayloadKHR-04699

IncomingRayPayloadKHR Storage Class **must** only be used in closest hit, any-hit, or miss shaders

- VUID-StandaloneSpirv-IncomingRayPayloadKHR-04700

There **must** be at most one variable with the **IncomingRayPayloadKHR Storage Class** in the input interface of an entry point

- VUID-StandaloneSpirv-HitAttributeKHR-04701

HitAttributeKHR Storage Class **must** only be used in intersection, any-hit, or closest hit shaders

- VUID-StandaloneSpirv-HitAttributeKHR-04702
There **must** be at most one variable with the **HitAttributeKHR Storage Class** in the input interface of an entry point
- VUID-StandaloneSpirv-HitAttributeKHR-04703
A variable with **HitAttributeKHR Storage Class** **must** only be written to in an intersection shader
- VUID-StandaloneSpirv-CallableDataKHR-04704
CallableDataKHR Storage Class **must** only be used in ray generation, closest hit, miss, and callable shaders
- VUID-StandaloneSpirv-IncomingCallableDataKHR-04705
IncomingCallableDataKHR Storage Class **must** only be used in callable shaders
- VUID-StandaloneSpirv-IncomingCallableDataKHR-04706
There **must** be at most one variable with the **IncomingCallableDataKHR Storage Class** in the input interface of an entry point
- VUID-StandaloneSpirv-ShaderRecordBufferKHR-07119
ShaderRecordBufferKHR Storage Class **must** only be used in ray generation, intersection, any-hit, closest hit, callable, or miss shaders
- VUID-StandaloneSpirv-Base-07650
The **Base** operand of **OpPtrAccessChain** or **OpUntypedPtrAccessChainKHR** **must** have a storage class of **Workgroup**, **StorageBuffer**, or **PhysicalStorageBuffer**
- VUID-StandaloneSpirv-Base-07651
If the **Base** operand of **OpPtrAccessChain** or **OpUntypedPtrAccessChainKHR** has a **Workgroup Storage Class**, then the **VariablePointers** capability **must** be declared
- VUID-StandaloneSpirv-Base-07652
If the **Base** operand of **OpPtrAccessChain** or **OpUntypedPtrAccessChainKHR** has a **StorageBuffer Storage Class**, then the **VariablePointers** or **VariablePointersStorageBuffer** capability **must** be declared
- VUID-StandaloneSpirv-OpUntypedVariableKHR-11167
Any **OpUntypedVariableKHR** **must** have a **Data Type** operand specified
- VUID-StandaloneSpirv-PhysicalStorageBuffer64-04708
If the **PhysicalStorageBuffer64** addressing model is enabled, all instructions that support memory access operands and that use a physical pointer **must** include the **Aligned** operand
- VUID-StandaloneSpirv-PhysicalStorageBuffer64-04709
If the **PhysicalStorageBuffer64** addressing model is enabled, any access chain instruction that accesses into a **RowMajor** matrix **must** only be used as the **Pointer** operand to **OpLoad** or **OpStore**
- VUID-StandaloneSpirv-PhysicalStorageBuffer64-04710
If the **PhysicalStorageBuffer64** addressing model is enabled, **OpConvertUToPtr** and **OpConvertPtrToU** **must** use an integer type whose **Width** is 64

- VUID-StandaloneSpirv-PhysicalStorageBuffer64-06314
If the **PhysicalStorageBuffer64** addressing model is enabled any load or store through a physical pointer type **must** have an **Aligned** operand which is a multiple of the size of the largest scalar type in the pointed-to type
- VUID-StandaloneSpirv-OpTypeForwardPointer-04711
OpTypeForwardPointer **must** have a **Storage Class** of **PhysicalStorageBuffer**
- VUID-StandaloneSpirv-None-04745
All block members in a variable with a **Storage Class** of **PushConstant** declared as an array **must** only be accessed by dynamically uniform indices
- VUID-StandaloneSpirv-OpVariable-06673
There **must** not be more than one variable in the **PushConstant Storage Class** listed in the **Interface** for each **OpEntryPoint**
- VUID-StandaloneSpirv-OpEntryPoint-06674
Each **OpEntryPoint** **must** not statically use more than one variable in the **PushConstant Storage Class**
- VUID-StandaloneSpirv-OpEntryPoint-08721
Each **OpEntryPoint** **must** not have more than one **Input** variable assigned the same **Component** word inside a **Location** slot, either explicitly or implicitly
- VUID-StandaloneSpirv-OpEntryPoint-08722
Each **OpEntryPoint** **must** not have more than one **Output** variable assigned the same **Component** word inside a **Location** slot, either explicitly or implicitly
- VUID-StandaloneSpirv-Result-04780
The **Result Type** operand of any **OpImageRead** or **OpImageSparseRead** instruction **must** be a vector of four components
- VUID-StandaloneSpirv-PushConstant-06675
Any variable in the **PushConstant** or **StorageBuffer** storage class **must** be decorated as **Block**
- VUID-StandaloneSpirv-Uniform-06676
Any variable in the **Uniform Storage Class** **must** be decorated as **Block** or **BufferBlock**
- VUID-StandaloneSpirv-UniformConstant-06677
Any variable in the **UniformConstant**, **StorageBuffer**, or **Uniform Storage Class** **must** be decorated with **DescriptorSet** and **Binding**
- VUID-StandaloneSpirv-InputAttachmentIndex-06678
Variables decorated with **InputAttachmentIndex** **must** be in the **UniformConstant Storage Class**
- VUID-StandaloneSpirv-DescriptorSet-06491
If a variable is decorated by **DescriptorSet** or **Binding**, the **Storage Class** **must** correspond to an entry in **Shader Resource and Storage Class Correspondence**
- VUID-StandaloneSpirv-Input-06778
Variables with a **Storage Class** of **Input** in a fragment shader stage that are decorated with **PerVertexKHR** **must** be declared as arrays
- VUID-StandaloneSpirv-MeshEXT-07102
The module **must** not contain both an entry point that uses the **TaskEXT** or **MeshEXT**

Execution Model and an entry point that uses the **TaskNV** or **MeshNV Execution Model**

- VUID-StandaloneSpirv-MeshEXT-07106
In mesh shaders using the **MeshEXT Execution Model** **OpSetMeshOutputsEXT** **must** be called before any outputs are written
- VUID-StandaloneSpirv-MeshEXT-07107
In mesh shaders using the **MeshEXT Execution Model** all variables declared as output **must** not be read from
- VUID-StandaloneSpirv-MeshEXT-07108
In mesh shaders using the **MeshEXT Execution Model** for **OpSetMeshOutputsEXT** instructions, the “Vertex Count” and “Primitive Count” operands **must** not depend on **ViewIndex**
- VUID-StandaloneSpirv-MeshEXT-07109
In mesh shaders using the **MeshEXT Execution Model** variables decorated with **PrimitivePointIndicesEXT**, **PrimitiveLineIndicesEXT**, or **PrimitiveTriangleIndicesEXT** declared as an array **must** not be accessed by indices that depend on **ViewIndex**
- VUID-StandaloneSpirv-MeshEXT-07110
In mesh shaders using the **MeshEXT Execution Model** any values stored in variables decorated with **PrimitivePointIndicesEXT**, **PrimitiveLineIndicesEXT**, or **PrimitiveTriangleIndicesEXT** **must** not depend on **ViewIndex**
- VUID-StandaloneSpirv-MeshEXT-07111
In mesh shaders using the **MeshEXT Execution Model** variables in workgroup or private **Storage Class** declared as or containing a composite type **must** not be accessed by indices that depend on **ViewIndex**
- VUID-StandaloneSpirv-MeshEXT-07330
In mesh shaders using the **MeshEXT Execution Model** the **OutputVertices Execution Mode** **must** be greater than 0
- VUID-StandaloneSpirv-MeshEXT-07331
In mesh shaders using the **MeshEXT Execution Model** the **OutputPrimitivesEXT Execution Mode** **must** be greater than 0
- VUID-StandaloneSpirv-Input-07290
Variables with a **Storage Class** of **Input** or **Output** and a type of **OpTypeBool** **must** be decorated with the **BuiltIn** decoration
- VUID-StandaloneSpirv-TileImageEXT-08723
The tile image variable declarations **must** obey the constraints on the **TileImageEXT Storage Class** and the **Location** decoration described in **Fragment Tile Image Interface**
- VUID-StandaloneSpirv-None-08724
The **TileImageEXT Storage Class** **must** only be used for declaring tile image variables
- VUID-StandaloneSpirv-Pointer-08973
The **Storage Class** of the **Pointer** operand to **OpCooperativeMatrixLoadKHR** or **OpCooperativeMatrixStoreKHR** **must** be limited to **Workgroup**, **StorageBuffer**, or **PhysicalStorageBuffer**
- VUID-StandaloneSpirv-OpTypeFloat-10370
Variables with a type of **OpTypeFloat** and an encoding of **BFloat16KHR** **must** not be declared with a **Storage Class** of **Input** or **Output**

- VUID-StandaloneSpirv-UniformBufferArrayDynamicIndexing-10127
If the **UniformBufferArrayDynamicIndexing** capability is not declared, and an instruction accesses memory through a uniform buffer, the uniform buffer through which that memory is accessed **must** be determined by **constant integral expressions**
- VUID-StandaloneSpirv-SampledImageArrayDynamicIndexing-10128
If the **SampledImageArrayDynamicIndexing** capability is not declared, and an instruction accesses memory through a sampled image or sampler, the sampled image or sampler through which that memory is accessed **must** be determined by **constant integral expressions**
- VUID-StandaloneSpirv-StorageBufferArrayDynamicIndexing-10129
If the **StorageBufferArrayDynamicIndexing** capability is not declared, and an instruction accesses memory through a storage buffer, the storage buffer through which that memory is accessed **must** be determined by **constant integral expressions**
- VUID-StandaloneSpirv-StorageImageArrayDynamicIndexing-10130
If the **StorageImageArrayDynamicIndexing** capability is not declared, and an instruction accesses memory through a storage image, the storage image through which that memory is accessed **must** be determined by **constant integral expressions**
- VUID-StandaloneSpirv-InputAttachmentArrayDynamicIndexing-10131
If the **InputAttachmentArrayDynamicIndexing** capability is not declared, and an instruction accesses memory through an input attachment, the input attachment through which that memory is accessed **must** be determined by **constant integral expressions**
- VUID-StandaloneSpirv-UniformTexelBufferArrayDynamicIndexing-10132
If the **UniformTexelBufferArrayDynamicIndexing** capability is not declared, and an instruction accesses memory through a uniform texel buffer, the uniform texel buffer through which that memory is accessed **must** be determined by **constant integral expressions**
- VUID-StandaloneSpirv-StorageTexelBufferArrayDynamicIndexing-10133
If the **StorageTexelBufferArrayDynamicIndexing** capability is not declared, and an instruction accesses memory through a storage texel buffer, the storage texel buffer through which that memory is accessed **must** be determined by **constant integral expressions**
- VUID-StandaloneSpirv-TileShadingQCOM-10686
TileShadingQCOM capability **must** not be enabled in any stage other than compute or fragment
- VUID-StandaloneSpirv-Execution-10687
Execution Mode NonCoherentTileAttachmentReadQCOM **must** not be used in any stage other than fragment
- VUID-StandaloneSpirv-Execution-10688
Execution Mode TileShadingRateQCOM **must** not be used in any stage other than compute
- VUID-StandaloneSpirv-TileAttachmentQCOM-10689
If the **TileAttachmentQCOM Storage Class** is used, the **TileShadingQCOM** capability **must** be enabled
- VUID-StandaloneSpirv-NonCoherentTileAttachmentReadQCOM-10690

If the `NonCoherentTileAttachmentReadQCOM` Execution Mode is used, the `TileShadingQCOM` capability **must** be enabled

- VUID-StandaloneSpirv-TileShadingRateQCOM-10691

If the `TileShadingRateQCOM` Execution Mode is used, the `TileShadingQCOM` capability **must** be enabled

- VUID-StandaloneSpirv-TileShadingRateQCOM-10692

If the `TileShadingRateQCOM` Execution Mode is used, `LocalSize` and `LocalSizeId` Execution Mode **must** not be specified

- VUID-StandaloneSpirv-OpTypeImage-10693

`OpTypeImage` variables in the `TileAttachmentQCOM` Storage Class **must** have `Dim` equal to 2D

- VUID-StandaloneSpirv-OpTypeImage-10694

`OpTypeImage` variables in the `TileAttachmentQCOM` Storage Class **must** `Sampled` equal to 1 or 2

- VUID-StandaloneSpirv-TileAttachmentQCOM-10695

Any variable in the `TileAttachmentQCOM` Storage Class **must** be decorated with `DescriptorSet` and `Binding`

- VUID-StandaloneSpirv-TileAttachmentQCOM-10696

Any variable in the `TileAttachmentQCOM` Storage Class **must** not be decorated with `Component` decoration

- VUID-StandaloneSpirv-TileAttachmentQCOM-10697

An `OpTypeImage` variables in the `TileAttachmentQCOM` Storage Class **must** not be consumed by an `OpImageQuery*` instruction

- VUID-StandaloneSpirv-OpTypeFloat-10823

Variables with a type of `OpTypeFloat` and an encoding of `Float8E4M3EXT` or `Float8E5M2EXT` **must** not be declared with a `Storage Class` of `Input` or `Output`

- VUID-StandaloneSpirv-OpGraphInputARM-09931

The `InputIndex` and `ElementIndex` operands to `OpGraphInputARM` **must** be the <id> of a constant instruction

- VUID-StandaloneSpirv-OpGraphSetOutputARM-09932

The `OutputIndex` and `ElementIndex` operands to `OpGraphSetOutputARM` **must** be the <id> of a constant instruction

Runtime SPIR-V Validation

The following rules **must** be validated at runtime. These rules depend on knowledge of the implementation and its capabilities and knowledge of runtime information, such as enabled features.

Valid Usage

- VUID-RuntimeSpirv-vulkanMemoryModel-06265

If the `vulkanMemoryModel` feature is enabled and the `vulkanMemoryModelDeviceScope` feature is not enabled, `Device` memory scope **must** not be used

- VUID-RuntimeSpirv-vulkanMemoryModel-06266
If the `vulkanMemoryModel` feature is not enabled, `QueueFamily` memory scope **must** not be used
- VUID-RuntimeSpirv-shaderSubgroupClock-06267
If the `shaderSubgroupClock` feature is not enabled, the `Subgroup` scope **must** not be used for `OpReadClockKHR`
- VUID-RuntimeSpirv-shaderDeviceClock-06268
If the `shaderDeviceClock` feature is not enabled, the `Device` scope **must** not be used for `OpReadClockKHR`
- VUID-RuntimeSpirv-OpTypeImage-06269
If the `shaderStorageImageWriteWithoutFormat` feature is not enabled, any variable created with a “Type” of `OpTypeImage` that has a “Sampled” operand of 2 and an “Image Format” operand of `Unknown` **must** be decorated with `NonWritable`
- VUID-RuntimeSpirv-OpTypeImage-06270
If the `shaderStorageImageReadWithoutFormat` feature is not enabled, any variable created with a “Type” of `OpTypeImage` that has a “Sampled” operand of 2 and an “Image Format” operand of `Unknown` **must** be decorated with `NonReadable`
- VUID-RuntimeSpirv-None-09558
Any variable created with a “Type” of `OpTypeImage` that has a “Dim” operand of `SubpassData` **must** be decorated with `InputAttachmentIndex`
- VUID-RuntimeSpirv-OpImageWrite-07112
`OpImageWrite` to any `Image` whose `Image Format` is not `Unknown` **must** have the `Texel` operand contain at least as many components as the corresponding `VkFormat` as given in the [SPIR-V Image Format compatibility table](#)
- VUID-RuntimeSpirv-Location-06272
The sum of `Location` and the number of locations the variable it decorates consumes **must** be less than or equal to the value for the matching `Execution Model` defined in [Shader Input and Output Locations](#)
- VUID-RuntimeSpirv-Location-06428
The maximum number of storage buffers, storage images, and output `Location` decorated color attachments written to in the `Fragment Execution Model` **must** be less than or equal to `maxFragmentCombinedOutputResources`
- VUID-RuntimeSpirv-UniformBufferArrayNonUniformIndexing-10134
If the `UniformBufferArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a uniform buffer, the uniform buffer through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup
- VUID-RuntimeSpirv-SampledImageArrayNonUniformIndexing-10135
If the `SampledImageArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a sampled image or sampler, the sampled image or sampler through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup
- VUID-RuntimeSpirv-StorageBufferArrayNonUniformIndexing-10136

If the `StorageBufferArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a storage buffer, the storage buffer through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup

- VUID-RuntimeSpirv-StorageImageArrayNonUniformIndexing-10137

If the `StorageImageArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a storage image, the storage image through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup

- VUID-RuntimeSpirv-InputAttachmentArrayNonUniformIndexing-10138

If the `InputAttachmentArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through an input attachment, the input attachment through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup

- VUID-RuntimeSpirv-UniformTexelBufferArrayNonUniformIndexing-10139

If the `UniformTexelBufferArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a uniform texel buffer, the uniform texel buffer through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup

- VUID-RuntimeSpirv-StorageTexelBufferArrayNonUniformIndexing-10140

If the `StorageTexelBufferArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a storage texel buffer, the storage texel buffer through which that memory is accessed **must** be dynamically uniform within the invocation group or subgroup

- VUID-RuntimeSpirv-subgroupSize-10141

If the `effective subgroup size` is 1, the `UniformBufferArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a uniform buffer, the uniform buffer through which that memory is accessed **must** be dynamically uniform within the invocation group

- VUID-RuntimeSpirv-subgroupSize-10142

If the `effective subgroup size` is 1, the `SampledImageArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a sampled image or sampler, the sampled image or sampler through which that memory is accessed **must** be dynamically uniform within the invocation group

- VUID-RuntimeSpirv-subgroupSize-10143

If the `effective subgroup size` is 1, the `StorageBufferArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a storage buffer, the storage buffer through which that memory is accessed **must** be dynamically uniform within the invocation group

- VUID-RuntimeSpirv-subgroupSize-10144

If the `effective subgroup size` is 1, the `StorageImageArrayNonUniformIndexing` capability is not declared, and an instruction accesses memory through a storage image, the storage image through which that memory is accessed **must** be dynamically uniform within the invocation group

- VUID-RuntimeSpirv-subgroupSize-10145

If the `effective subgroup size` is 1, the `InputAttachmentArrayNonUniformIndexing` capability is

not declared, and an instruction accesses memory through an input attachment, the input attachment through which that memory is accessed **must** be dynamically uniform within the invocation group

- VUID-RuntimeSpirv-subgroupSize-10146

If the **effective subgroup size** is 1, the **UniformTexelBufferArrayNonUniformIndexing** capability is not declared, and an instruction accesses memory through a uniform texel buffer, the uniform texel buffer through which that memory is accessed **must** be dynamically uniform within the invocation group

- VUID-RuntimeSpirv-subgroupSize-10147

If the **effective subgroup size** is 1, the **StorageTexelBufferArrayNonUniformIndexing** capability is not is not declared, and an instruction accesses memory through a storage texel buffer, the storage texel buffer through which that memory is accessed **must** be dynamically uniform within the invocation group

- VUID-RuntimeSpirv-None-10148

If an instruction accesses memory through any resource, the **effective subgroup size** is 1, and the resource through which that memory is accessed is not uniform within the invocation group, then the operand corresponding to that resource (e.g. the pointer or sampled image operand) **must** be decorated with **NonUniform**

- VUID-RuntimeSpirv-subgroupSize-10149

If an instruction accesses memory through any resource, the **effective subgroup size** is greater than 1, and the resource through which that memory is accessed is not uniform within the invocation group, and not uniform within the subgroup, then the operand corresponding to that resource (e.g. the pointer or sampled image operand) **must** be decorated with **NonUniform**

- VUID-RuntimeSpirv-None-06275

shaderSubgroupExtendedTypes **must** be enabled for **group operations** to use 8-bit integer, 16-bit integer, 64-bit integer, 16-bit floating-point, and vectors of these types

- VUID-RuntimeSpirv-subgroupBroadcastDynamicId-06276

If **subgroupBroadcastDynamicId** is **VK_TRUE**, and the shader module version is 1.5 or higher, the “Index” for **OpGroupNonUniformQuadBroadcast** **must** be dynamically uniform within the derivative group. Otherwise, “Index” **must** be a constant

- VUID-RuntimeSpirv-subgroupBroadcastDynamicId-06277

If **subgroupBroadcastDynamicId** is **VK_TRUE**, and the shader module version is 1.5 or higher, the “Id” for **OpGroupNonUniformBroadcast** **must** be dynamically uniform within the subgroup. Otherwise, “Id” **must** be a constant

- VUID-RuntimeSpirv-None-06284

shaderBufferFloat32Atomics, or **shaderBufferFloat32AtomicAdd**, or **shaderBufferFloat64Atomics**, or **shaderBufferFloat64AtomicAdd**, **must** be enabled for floating-point atomic operations to be supported on a *Pointer* with a **Storage Class** of **StorageBuffer**

- VUID-RuntimeSpirv-None-06285

shaderSharedFloat32Atomics, or **shaderSharedFloat32AtomicAdd**, or **shaderSharedFloat64Atomics**, or **shaderSharedFloat64AtomicAdd**, **must** be enabled for floating-point atomic operations to be supported on a *Pointer* with a **Storage Class** of

Workgroup

- VUID-RuntimeSpirv-None-06286
`shaderImageFloat32Atomics`, or `shaderImageFloat32AtomicAdd`, **must** be enabled for 32-bit floating-point atomic operations to be supported on a *Pointer* with a *Storage Class* of *Image*
- VUID-RuntimeSpirv-None-06287
`sparseImageFloat32Atomics`, or `sparseImageFloat32AtomicAdd`, **must** be enabled for 32-bit floating-point atomics to be supported on sparse images
- VUID-RuntimeSpirv-None-06288
`shaderImageInt64Atomics` **must** be enabled for 64-bit integer atomic operations to be supported on a *Pointer* with a *Storage Class* of *Image*
- VUID-RuntimeSpirv-denormBehaviorIndependence-06289
If `denormBehaviorIndependence` is `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_32_BIT_ONLY`, then the entry point **must** use the same denormals *Execution Mode* for both 16-bit and 64-bit floating-point types
- VUID-RuntimeSpirv-denormBehaviorIndependence-06290
If `denormBehaviorIndependence` is `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_NONE`, then the entry point **must** use the same denormals *Execution Mode* for all floating-point types
- VUID-RuntimeSpirv-roundingModeIndependence-06291
If `roundingModeIndependence` is `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_32_BIT_ONLY`, then the entry point **must** use the same rounding *Execution Mode* for both 16-bit and 64-bit floating-point types
- VUID-RuntimeSpirv-roundingModeIndependence-06292
If `roundingModeIndependence` is `VK_SHADER_FLOAT_CONTROLS_INDEPENDENCE_NONE`, then the entry point **must** use the same rounding *Execution Mode* for all floating-point types
- VUID-RuntimeSpirv-shaderSignedZeroInfNanPreserveFloat16-06293
If `shaderSignedZeroInfNanPreserveFloat16` is `VK_FALSE`, then *SignedZeroInfNanPreserve* for 16-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderSignedZeroInfNanPreserveFloat32-06294
If `shaderSignedZeroInfNanPreserveFloat32` is `VK_FALSE`, then *SignedZeroInfNanPreserve* for 32-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderSignedZeroInfNanPreserveFloat64-06295
If `shaderSignedZeroInfNanPreserveFloat64` is `VK_FALSE`, then *SignedZeroInfNanPreserve* for 64-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderDenormPreserveFloat16-06296
If `shaderDenormPreserveFloat16` is `VK_FALSE`, then *DenormPreserve* for 16-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderDenormPreserveFloat32-06297
If `shaderDenormPreserveFloat32` is `VK_FALSE`, then *DenormPreserve* for 32-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderDenormPreserveFloat64-06298
If `shaderDenormPreserveFloat64` is `VK_FALSE`, then *DenormPreserve* for 64-bit floating-point type **must** not be used

- VUID-RuntimeSpirv-shaderDenormFlushToZeroFloat16-06299
If `shaderDenormFlushToZeroFloat16` is `VK_FALSE`, then `DenormFlushToZero` for 16-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderDenormFlushToZeroFloat32-06300
If `shaderDenormFlushToZeroFloat32` is `VK_FALSE`, then `DenormFlushToZero` for 32-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderDenormFlushToZeroFloat64-06301
If `shaderDenormFlushToZeroFloat64` is `VK_FALSE`, then `DenormFlushToZero` for 64-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderRoundingModeRTEFloat16-06302
If `shaderRoundingModeRTEFloat16` is `VK_FALSE`, then `RoundingModeRTE` for 16-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderRoundingModeRTEFloat32-06303
If `shaderRoundingModeRTEFloat32` is `VK_FALSE`, then `RoundingModeRTE` for 32-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderRoundingModeRTEFloat64-06304
If `shaderRoundingModeRTEFloat64` is `VK_FALSE`, then `RoundingModeRTE` for 64-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderRoundingModeRTZFloat16-06305
If `shaderRoundingModeRTZFloat16` is `VK_FALSE`, then `RoundingModeRTZ` for 16-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderRoundingModeRTZFloat32-06306
If `shaderRoundingModeRTZFloat32` is `VK_FALSE`, then `RoundingModeRTZ` for 32-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-shaderRoundingModeRTZFloat64-06307
If `shaderRoundingModeRTZFloat64` is `VK_FALSE`, then `RoundingModeRTZ` for 64-bit floating-point type **must** not be used
- VUID-RuntimeSpirv-PhysicalStorageBuffer64-06315
If the `PhysicalStorageBuffer64` addressing model is enabled the pointer value of a memory access instruction **must** be at least as aligned as specified by the `Aligned` memory access operand
- VUID-RuntimeSpirv-None-10980
Objects containing 8-bit integer or 8-bit floating-point elements **must** not have storage class of **Input** or **Output**
- VUID-RuntimeSpirv-None-06335
`shaderBufferFloat32Atomics`, or `shaderBufferFloat32AtomicAdd`, or `shaderSharedFloat32Atomics`, or `shaderSharedFloat32AtomicAdd`, or `shaderImageFloat32Atomics`, or `shaderImageFloat32AtomicAdd` **must** be enabled for 32-bit floating-point atomic operations
- VUID-RuntimeSpirv-None-06336
`shaderBufferFloat64Atomics`, or `shaderBufferFloat64AtomicAdd`, or `shaderSharedFloat64Atomics`, or `shaderSharedFloat64AtomicAdd` **must** be enabled for 64-bit floating-point atomic operations

- VUID-RuntimeSpirv-NonWritable-06340
If the `fragmentStoresAndAtomics` feature is not enabled, then all storage image, storage texel buffer, and storage buffer variables in the fragment stage **must** be decorated with the `NonWritable` decoration
- VUID-RuntimeSpirv-NonWritable-06341
If the `vertexPipelineStoresAndAtomics` feature is not enabled, then all storage image, storage texel buffer, and storage buffer variables in the vertex, tessellation, and geometry stages **must** be decorated with the `NonWritable` decoration
- VUID-RuntimeSpirv-OpAtomic-05091
If the `shaderAtomicInstructions` feature is not enabled, the SPIR-V Atomic Instructions listed in 3.37.18 (`OpAtomic*`) **must** not be used [SCID-1]
- VUID-RuntimeSpirv-None-06342
If `subgroupQuadOperationsInAllStages` is `VK_FALSE`, then `quad subgroup operations` **must** not be used except for in fragment and compute stages
- VUID-RuntimeSpirv-None-06343
`Group operations` with `subgroup scope` **must** not be used if the shader stage is not in `subgroupSupportedStages`
- VUID-RuntimeSpirv-Offset-06344
The first element of the `Offset` operand of `InterpolateAtOffset` **must** be greater than or equal to:
$$\text{frag}_{\text{width}} \times \text{minInterpolationOffset}$$

where $\text{frag}_{\text{width}}$ is the width of the current fragment in pixels
- VUID-RuntimeSpirv-Offset-06345
The first element of the `Offset` operand of `InterpolateAtOffset` **must** be less than or equal to
$$\text{frag}_{\text{width}} \times (\text{maxInterpolationOffset} + \text{ULP}) - \text{ULP}$$

where $\text{frag}_{\text{width}}$ is the width of the current fragment in pixels and $\text{ULP} = 1 / 2^{\text{subPixelInterpolationOffsetBits}}$
- VUID-RuntimeSpirv-Offset-06346
The second element of the `Offset` operand of `InterpolateAtOffset` **must** be greater than or equal to
$$\text{frag}_{\text{height}} \times \text{minInterpolationOffset}$$

where $\text{frag}_{\text{height}}$ is the height of the current fragment in pixels
- VUID-RuntimeSpirv-Offset-06347
The second element of the `Offset` operand of `InterpolateAtOffset` **must** be less than or equal to
$$\text{frag}_{\text{height}} \times (\text{maxInterpolationOffset} + \text{ULP}) - \text{ULP}$$

where $\text{frag}_{\text{height}}$ is the height of the current fragment in pixels and $\text{ULP} = 1 / 2^{\text{subPixelInterpolationOffsetBits}}$
- VUID-RuntimeSpirv-x-06429
In compute shaders using the `GLCompute Execution Model` the `x` size in `LocalSize` or `LocalSizeId` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupSize[0]`
- VUID-RuntimeSpirv-y-06430

In compute shaders using the `GLCompute Execution Model` the `y` size in `LocalSize` or `LocalSizeId` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupSize[1]`

- VUID-RuntimeSpirv-z-06431

In compute shaders using the `GLCompute Execution Model` the `z` size in `LocalSize` or `LocalSizeId` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupSize[2]`

- VUID-RuntimeSpirv-x-06432

In compute shaders using the `GLCompute Execution Model` the product of `x` size, `y` size, and `z` size in `LocalSize` or `LocalSizeId` **must** be less than or equal to `VkPhysicalDeviceLimits::maxComputeWorkGroupInvocations`

- VUID-RuntimeSpirv-LocalSizeId-06433

The `Execution Mode LocalSizeId` **must** not be used

- VUID-RuntimeSpirv-OpTypeVector-06816

Any `OpTypeVector` output interface variables **must** not have a higher `Component Count` than a matching `OpTypeVector` input interface variable

- VUID-RuntimeSpirv-OpEntryPoint-08743

Any `user-defined variables` shared between the `OpEntryPoint` of two shader stages, and declared with `Input` as its `Storage Class` for the subsequent shader stage, **must** have all `Location` slots and `Component` words declared in the preceding shader stage's `OpEntryPoint` with `Output` as the `Storage Class`

- VUID-RuntimeSpirv-OpEntryPoint-07754

Any `user-defined variables` between the `OpEntryPoint` of two shader stages **must** have the same type and width for each `Component`

- VUID-RuntimeSpirv-OpVariable-08746

Any variable, `Block`-decorated `OpTypeStruct`, or `Block`-decorated `OpTypeStruct` members shared between the `OpEntryPoint` of two shader stages **must** have matching decorations as defined in `interface matching`

- VUID-RuntimeSpirv-Workgroup-06530

The sum of size in bytes for variables and `padding` in the `Workgroup Storage Class` in the `GLCompute Execution Model` **must** be less than or equal to `maxComputeSharedMemorySize`

- VUID-RuntimeSpirv-OpVariable-06373

Any variable with `Workgroup` as its `Storage Class` **must** not have an `Initializer` operand

- VUID-RuntimeSpirv-Offset-10213

Image operand `Offset` **must** only be used with `OpImage*Gather` instructions

- VUID-RuntimeSpirv-OpImage-06376

If an `OpImage*Gather` operation has an image operand of `Offset`, `ConstOffset`, or `ConstOffsets` the offset value **must** be greater than or equal to `minTexelGatherOffset`

- VUID-RuntimeSpirv-OpImage-06377

If an `OpImage*Gather` operation has an image operand of `Offset`, `ConstOffset`, or `ConstOffsets` the offset value **must** be less than or equal to `maxTexelGatherOffset`

- VUID-RuntimeSpirv-OpImageSample-06435

If an `OpImageSample*` or `OpImageFetch*` operation has an image operand of `ConstOffset` then

the offset value **must** be greater than or equal to `minTexelOffset`

- VUID-RuntimeSpirv-OpImageSample-06436

If an `OpImageSample*` or `OpImageFetch*` operation has an image operand of `ConstOffset` then the offset value **must** be less than or equal to `maxTexelOffset`

- VUID-RuntimeSpirv-samples-08725

If an `OpTypeImage` has an `MS` operand 0, its bound image **must** have been created with `VkImageCreateInfo::samples` as `VK_SAMPLE_COUNT_1_BIT`

- VUID-RuntimeSpirv-samples-08726

If an `OpTypeImage` has an `MS` operand 1, its bound image **must** not have been created with `VkImageCreateInfo::samples` as `VK_SAMPLE_COUNT_1_BIT`

- VUID-RuntimeSpirv-OpEntryPoint-08727

Each `OpEntryPoint` **must** not have more than one variable decorated with `InputAttachmentIndex` per image aspect of the attachment image bound to it, either explicitly or implicitly as described by [input attachment interface](#)

- VUID-RuntimeSpirv-MeshEXT-09218

In mesh shaders using the `MeshEXT` or `MeshNV Execution Model` and the `OutputPoints Execution Mode`, if the number of output points is greater than 0, a `PointSize` decorated variable **must** be written to for each output point

- VUID-RuntimeSpirv-protectedNoFault-09645

If `protectedNoFault` is not supported, the `Storage Class` of the `PhysicalStorageBuffer` **must** not be used if the buffer being accessed is `protected`

- VUID-RuntimeSpirv-None-10715

If a `VkSampler` object that enables `sampler YBCR conversion` is accessed in a shader, both the sampler and image accessed **must** be determined by [constant integral expressions](#)

- VUID-RuntimeSpirv-None-10716

If a `VkSampler` object that enables `sampler YBCR conversion` is statically used, it **must** only be used with `OpImageSample*` or `OpImageSparseSample*` instructions

- VUID-RuntimeSpirv-OpImage-10717

An `OpImage` instruction **must** not be used on a `VkSampler` object that enables `sampler YBCR conversion`

- VUID-RuntimeSpirv-ConstOffset-10718

If a `VkSampler` object that enables `sampler YBCR conversion` is statically used, it **must** not use the `ConstOffset` or `Offset` operands

- VUID-RuntimeSpirv-None-10824

The `Base` operand of any `OpBitCount`, `OpBitReverse`, `OpBitFieldInsert`, `OpBitFieldSExtract`, or `OpBitFieldUExtract` instruction **must** be a 32-bit integer scalar or a vector of 32-bit integers

- VUID-RuntimeSpirv-None-10834

[Buffer indexing calculations](#) **must** not wrap 32 bits

- VUID-RuntimeSpirv-OpArrayLength-11807

`OpArrayLength` result type **must** be a 32-bit integer type

Precision and Operation of SPIR-V Instructions

The following rules apply to operations on all floating-point values:

- Positive and negative infinities and positive and negative zeros are generated as dictated by [IEEE 754](#) for the specified encoding, but subject to the precisions allowed by each operation for a given encoding, as specified in [Precision of Individual Operations](#).
- Signaling NaNs are not required to be generated and exceptions are never raised. Signaling NaN **may** be converted to quiet NaNs values by any floating-point instruction.
- The set of operations `OpPhi`, `OpSelect`, `OpFunctionCall`, `OpReturnValue`, `OpVectorExtractDynamic`, `OpVectorInsertDynamic`, `OpVectorShuffle`, `OpCompositeConstruct`, `OpCompositeExtract`, `OpCompositeInsert`, `OpTranspose`, `OpCopyObject`, `OpCopyLogical`, `OpCopyMemory`, `OpGroupNonUniformBroadcast`, `OpGroupNonUniformBroadcastFirst`, `OpGroupNonUniformShuffle`, `OpGroupNonUniformShuffleXor`, `OpGroupNonUniformShuffleUp`, `OpGroupNonUniformShuffleDown`, `OpGroupNonUniformQuadBroadcast`, `OpGroupNonUniformQuadSwap`, `OpAtomicLoad`, `OpAtomicStore`, `OpAtomicExchange`, `OpStore`, and `OpLoad` are referred to as *bit-preserving operations*.
- By default, the implementation **may** perform optimizations on half, single, or double-precision floating-point instructions that ignore sign of a zero, or assume that arguments and results are not NaNs or infinities. If the entry point is declared with the `SignedZeroInfNanPreserve Execution Mode`, then NaNs, infinities, and the sign of zero **must** not be ignored.
 - All bit-preserving operations except `OpLoad` from the `Input Storage Class` in the fragment shader stage **must** respect the `SignedZeroInfNanPreserve Execution Mode`.
 - The following core SPIR-V instructions **must** respect the `SignedZeroInfNanPreserve Execution Mode`: `OpFConvert`, `OpFNegate`, `OpFAdd`, `OpFSub`, `OpFMul`, `OpFDiv`, `OpIsNan`, `OpIsInf`, `OpVectorTimesScalar`, `OpMatrixTimesScalar`, `OpVectorTimesMatrix`, `OpMatrixTimesVector`, `OpMatrixTimesMatrix`, `OpOuterProduct`, `OpDot`, `OpFOrdEqual`, `OpFUnordEqual`, `OpFOrdNotEqual`, `OpFUnordNotEqual`, `OpFOrdLessThan`, `OpFUnordLessThan`, `OpFOrdGreaterThan`, `OpFUnordGreaterThan`, `OpFOrdLessThanEqual`, `OpFUnordLessThanEqual`, `OpFOrdGreaterThanEqual`, `OpFUnordGreaterThanEqual`, `OpGroupNonUniformAlIEqual`, `OpGroupNonUniformFMin`, `OpGroupNonUniformFMax`, `OpAtomicCompareExchange`, `OpAtomicCompareExchangeWeak`, `OpDPdx`, `OpDPdy`, `OpFwidth`, `OpDPdxFine`, `OpDPdyFine`, `OpFwidthFine`, `OpDPdxCoarse`, `OpDPdyCoarse`, and `OpFwidthCoarse`.
- All bit-preserving operations and the following instructions **must** not flush denormalized values: `OpConstant`, `OpConstantComposite`, `OpSpecConstant`, `OpSpecConstantComposite`, and `OpBitcast`.
- Denormalized values are supported.
 - By default any denormalized floating-point value input into a shader or potentially generated by any instruction (except those listed above) or any extended instructions for GLSL in a shader **may** be flushed to zero.
 - If the entry point is declared with the `DenormFlushToZero Execution Mode` then for the affected instructions the denormalized result **must** be flushed to zero and the denormalized operands **may** be flushed to zero. Denormalized values obtained via unpacking an integer into a vector of values with smaller bit width and interpreting those values as floating-point numbers **must** be flushed to zero.
 - When denormal values are being flushed, the result of an operation **may** be considered

denormal whenever the infinitely precise result is non-zero and of smaller magnitude than the smallest normal value, even if rounding would otherwise give a normal result.

- The following core SPIR-V instructions **must** respect the **DenormFlushToZero Execution Mode**: **OpSpecConstantOp** (with opcode **OpFConvert**), **OpFConvert**, **OpFNegate**, **OpFAdd**, **OpFSub**, **OpFMul**, **OpFDiv**, **OpFRem**, **OpFMod**, **OpVectorTimesScalar**, **OpMatrixTimesScalar**, **OpVectorTimesMatrix**, **OpMatrixTimesVector**, **OpMatrixTimesMatrix**, **OpOuterProduct**, **OpDot**, **OpGroupNonUniformFMin**, **OpGroupNonUniformFMax**, **OpAtomicFAddEXT**, **OpDPdx**, **OpDPdy**, **OpFwidth**, **OpDPdxFine**, **OpDPdyFine**, **OpFwidthFine**, **OpDPdxCoarse**, **OpDPdyCoarse**, **OpFwidthCoarse**; and the following extended instructions for GLSL: **Round**, **RoundEven**, **Trunc**, **FAbs**, **Floor**, **Ceil**, **Fract**, **Radians**, **Degrees**, **Sin**, **Cos**, **Tan**, **Asin**, **Acos**, **Atan**, **Sinh**, **Cosh**, **Tanh**, **Asinh**, **Acosh**, **Atanh**, **Atan2**, **Pow**, **Exp**, **Log**, **Exp2**, **Log2**, **Sqrt**, **InverseSqrt**, **Determinant**, **MatrixInverse**, **Modf**, **ModfStruct**, **FMin**, **FMax**, **FClamp**, **FMix**, **Step**, **SmoothStep**, **Fma**, **UnpackHalf2x16**, **Length**, **Distance**, **Cross**, **Normalize**, **FaceForward**, **Reflect**, **Refract**, **NMin**, **NMax**, and **NClamp**.
- The following core SPIR-V instructions **must** respect the **DenormPreserve Execution Mode** for floating-point values with an **IEEE 754** encoding: **OpSpecConstantOp**, **OpFConvert**, **OpFNegate**, **OpFAdd**, **OpFSub**, **OpFMul**, **OpVectorTimesScalar**, **OpMatrixTimesScalar**, **OpVectorTimesMatrix**, **OpMatrixTimesVector**, **OpMatrixTimesMatrix**, **OpOuterProduct**, **OpDot**, **OpFOrdEqual**, **OpFUnordEqual**, **OpFOrdNotEqual**, **OpFUnordNotEqual**, **OpFOrdLessThan**, **OpFUnordLessThan**, **OpFOrdGreaterThan**, **OpFUnordGreaterThan**, **OpFOrdLessThanEqual**, **OpFUnordLessThanEqual**, **OpFOrdGreaterThanEqual**, **OpFUnordGreaterThanEqual**, **OpGroupNonUniformAllEqual**, **OpGroupNonUniformFMin**, **OpGroupNonUniformFMax**, **OpAtomicCompareExchange**, **OpAtomicCompareExchangeWeak**, **OpAtomicFAddEXT**, **OpDPdx**, **OpDPdy**, **OpFwidth**, **OpDPdxFine**, **OpDPdyFine**, **OpFwidthFine**, **OpDPdxCoarse**, **OpDPdyCoarse**, **OpFwidthCoarse**; and the following extended instructions for GLSL: **FAbs**, **FSign**, **Radians**, **Degrees**, **FMin**, **FMax**, **FClamp**, **FMix**, **Fma**, **PackHalf2x16**, **PackDouble2x32**, **UnpackHalf2x16**, **UnpackDouble2x32**, **NMin**, **NMax**, and **NClamp**.

The precision of double-precision instructions is at least that of single precision.

The precision of individual operations is defined in **Precision of Individual Operations**. Subject to the constraints below, however, implementations **may** reorder or combine operations, resulting in expressions exhibiting different precisions than might be expected from the constituent operations.

Evaluation of Expressions

Implementations **may** rearrange floating-point operations using any of the mathematical properties governing the expressions in precise arithmetic, even where the floating-point operations do not share these properties. This includes, but is not limited to, associativity and distributivity, and **may** involve a different number of rounding steps than would occur if the operations were not rearranged. In shaders that use the **SignedZeroInfNanPreserve Execution Mode** the values **must** be preserved if they are generated after any rearrangement but the **Execution Mode** does not change which rearrangements are valid. This rearrangement **can** be prevented for particular operations by using the **NoContraction** decoration.

NOTE

For example, in the absence of the **NoContraction** decoration implementations are allowed to implement $a + b - a$ and $\frac{a \times b}{a}$ as b . The **SignedZeroInfNanPreserve** does not prevent these transformations, even though they may overflow to infinity or NaN when evaluated in floating-point.

If the **NoContraction** decoration is applied then operations may not be rearranged, so, for example, $a + a - a$ must account for possible overflow to infinity. If infinities are not preserved then the expression may be replaced with a , since the replacement is exact when overflow does not occur and infinities may be replaced with undefined values. If both **NoContraction** and **SignedZeroInfNanPreserve** are used then the result must be infinity for sufficiently large a .

Precision of Individual Operations

The precision of individual operations is defined either in terms of rounding (correctly rounded), as an error bound in ULP, or as inherited from a formula as follows:

Correct Result

Operations that are described as returning the “correct result” will return the infinitely precise result which, due to the nature of the operation, will not need rounding.

Correctly Rounded

Operations described as “correctly rounded” will return the infinitely precise result, x , rounded so as to be representable in floating-point. If the entry point is declared with the **RoundingModeRTE** or the **RoundingModeRTZ Execution Mode** then this is done according to [IEEE 754](#) “roundTiesToEven” or “roundTowardZero” rounding directions, respectively.

Otherwise, they are rounded with [implementation-defined rounding mode](#).

Correctly Rounded with Implementation-Defined Rounding Mode

Operations described as “correctly rounded with implementation-defined rounding mode” will return the infinitely precise result, x , rounded so as to be representable in floating-point. If x is exactly representable then x will be returned. Otherwise, either the floating-point value closest to and no less than x or the value closest to and no greater than x will be returned. Which value is chosen is implementation-defined.

ULP

Where an error bound of n ULP (units in the last place) is given, for an operation with infinitely precise result x the value returned **must** be in the range $[x - n \times \text{ulp}(x), x + n \times \text{ulp}(x)]$. The function $\text{ulp}(x)$ is defined as follows:

If there exist non-equal, finite floating-point numbers a and b such that $a \leq x \leq b$ then $\text{ulp}(x)$ is the minimum possible distance between such numbers, $\text{ulp}(x) = \min_{a, b} |b - a|$. If such numbers do not exist then $\text{ulp}(x)$ is defined to be the difference between the two non-equal, finite floating-point numbers nearest to x .

Where the range of allowed return values includes any value of magnitude larger than that of the largest representable finite floating-point number, operations **may**, additionally, return either an infinity of the appropriate sign or the finite number with the largest magnitude of the appropriate sign. If the infinitely precise result of the operation is not mathematically defined then the value returned is undefined.

Inherited From ...

Where an operation's precision is described as being inherited from a formula, the result returned **must** be at least as accurate as the result of computing an approximation to x using a formula equivalent to the given formula applied to the supplied inputs. Specifically, the formula given may be transformed using the mathematical associativity, commutativity and distributivity of the operators involved to yield an equivalent formula. The SPIR-V precision rules, when applied to each such formula and the given input values, define a range of permitted values. If NaN is one of the permitted values then the operation may return any result, otherwise let the largest permitted value in any of the ranges be $F_{\max}$ and the smallest be $F_{\min}$. The operation **must** return a value in the range $[x - E, x + E]$ where $E = \max\{\tilde{A}|x - F_{\min}|, |x - F_{\max}|\}$. If the entry point is declared with the **DenormPreserve Execution Mode**, then denormals **must** be preserved throughout the formula. Otherwise, any intermediate denormal value(s) while evaluating the formula **may** be flushed to zero. If the entry point is declared with the **DenormFlushToZero** execution mode, then denormal final results **must** be flushed to zero.

For IEEE754 half- (16 bit) and single- (32 bit) precision instructions, precisions are **required** to be at least as follows:

Table 81. Precision of Core SPIR-V Instructions

Instruction	Single precision, unless decorated with RelaxedPrecision	Half precision
OpFNegate	Correct result.	
OpFAdd	Correctly rounded.	
OpFSub	Correctly rounded.	
OpFMul, OpVectorTimesScalar, OpMatrixTimesScalar	Correctly rounded.	
OpMatrixTimesVector	Inherited from $X_{j=0}^{n-1} M_{ij} \times v_j$.	
OpVectorTimesMatrix	Inherited from $X_{j=0}^{n-1} v_j \times M_{ji}$.	
OpMatrixTimesMatrix	Inherited from $X_{k=0}^{n-1} M_{ik} \times N_{kj}$.	
OpOuterProduct	Correctly rounded.	
OpFmaKHR	Correctly rounded.	
OpDot(x, y)	Inherited from $X_{i=0}^{n-1} x_i \times y_i$.	
OpIsNan, OpIsInf	Correct result.	
OpFOrdEqual, OpFUnordEqual	Correct result.	
OpFOrdNotEqual, OpFUnordNotEqual	Correct result.	
OpFOrdLessThan, OpFUnordLessThan	Correct result.	
OpFOrdGreaterThan, OpFUnordGreaterThan	Correct result.	

Instruction	Single precision, unless decorated with RelaxedPrecision	Half precision
OpFOrdLessThanEqual, OpFUnordLessThanEqual	Correct result.	
OpFOrdGreaterThanEqual, OpFUnordGreaterThanEqual	Correct result.	
OpGroupNonUniformAllEqual	Correct result.	
OpGroupNonUniformFMin, OpGroupNonUniformFMax	Correct result.	
OpFDiv(x,y)	2.5 ULP for $ y = 0$ or $ y $ in the range $[2^{-126}, 2^{126}]$.	2.5 ULP for $ y = 0$ or $ y $ in the range $[2^{-14}, 2^{14}]$.
OpFRem(x,y)	Inherited from $x - y \times \text{trunc}(x/y)$.	
OpFMod(x,y)	Inherited from $x - y \times \text{floor}(x/y)$.	
OpQuantizeToF16	Correctly rounded with implementation defined rounding mode.	
conversions between types	Correctly rounded.	
OpAtomicCompareExchange, OpAtomicCompareExchangeWeak	Correct result.	
OpAtomicFAddEXT	Return value correct result, value in memory correctly rounded.	
OpDPdx, OpDPdy, OpFwidth	Correctly rounded.	
OpDPdxCoarse, OpDPdyCoarse, OpFwidthCoarse	Correctly rounded.	
OpDPdxFine, OpDPdyFine, OpFwidthFine	Correctly rounded.	

NOTE

The **OpFRem** and **OpFMod** instructions use cheap approximations of remainder, and the error can be large due to the discontinuity in `trunc()` and `floor()`. This can produce mathematically unexpected results in some cases, such as `FMod(x,x)` computing `x` rather than `0`, and can also cause the result to have a different sign than the infinitely precise result.

Table 82. Precision of GLSL.std.450 Instructions

Instruction	Single precision, unless decorated with RelaxedPrecision	Half precision
fma()	Inherited from OpFMul followed by OpFAdd .	
exp(x), exp2(x)	$3 + 2 \times x $ ULP.	$1 + 2 \times x $ ULP.
log(), log2()	3 ULP outside the range $[0.5, 2.0]$. Absolute error $< 2^{-21}$ inside the range $[0.5, 2.0]$.	3 ULP outside the range $[0.5, 2.0]$. Absolute error $< 2^{-7}$ inside the range $[0.5, 2.0]$.
pow(x, y)	Inherited from $\text{exp2}(y \times \text{log2}(x))$.	

Instruction	Single precision, unless decorated with RelaxedPrecision	Half precision
<code>sqrt()</code>	Inherited from 1.0 / <code>inversesqrt()</code> .	
<code>inversesqrt()</code>	2 ULP.	
<code>radians(x)</code>	Inherited from $x \times C_{\pi\_180}$, where C_{π_180} is a correctly rounded approximation to $\frac{\pi}{180}$.	
<code>degrees(x)</code>	Inherited from $x \times C_{180\_pi}$, where C_{180_pi} is a correctly rounded approximation to $\frac{180}{\pi}$.	
<code>sin()</code>	Absolute error $\leq 2^{-11}$ inside the range $[-\pi, \pi]$.	Absolute error $\leq 2^{-7}$ inside the range $[-\pi, \pi]$.
<code>cos()</code>	Absolute error $\leq 2^{-11}$ inside the range $[-\pi, \pi]$.	Absolute error $\leq 2^{-7}$ inside the range $[-\pi, \pi]$.
<code>tan()</code>	Inherited from $\frac{\sin}{\cos}$.	
<code>asin(x)</code>	Inherited from <code>atan2(x, sqrt(1.0 - x * x))</code> .	
<code>acos(x)</code>	Inherited from <code>atan2(sqrt(1.0 - x * x), x)</code> .	
<code>atan(), atan2()</code>	4096 ULP	5 ULP.
<code>sinh(x)</code>	Inherited from $(\exp(x) - \exp(-x)) \times 0.5$.	
<code>cosh(x)</code>	Inherited from $(\exp(x) + \exp(-x)) \times 0.5$.	
<code>tanh()</code>	Inherited from $\frac{\sinh}{\cosh}$.	
<code>asinh(x)</code>	Inherited from <code>log(x + sqrt(x * x + 1.0))</code> .	
<code>acosh(x)</code>	Inherited from <code>log(x + sqrt(x * x - 1.0))</code> .	
<code>atanh(x)</code>	Inherited from <code>log((1.0 + x) / (1.0 - x))</code> $\times 0.5$.	
<code>frexp()</code>	Correct result.	
<code>ldexp()</code>	Correctly rounded.	
<code>length(x)</code>	Inherited from <code>sqrt(dot(x, x))</code> .	
<code>distance(x, y)</code>	Inherited from <code>length(x - y)</code> .	
<code>cross()</code>	Inherited from <code>OpFSub(OpFMul, OpFMul)</code> .	
<code>normalize(x)</code>	Inherited from <code>x * inversesqrt(dot(x, x))</code> .	
<code>faceforward(N, I, NRef)</code>	Inherited from <code>dot(NRef, I) < 0.0 ? N : -N</code> .	
<code>reflect(x, y)</code>	Inherited from <code>x - 2.0 * dot(y, x) * y</code> .	
<code>refract(I, N, eta)</code>	Inherited from <code>k < 0.0 ? 0.0 : eta * I - (eta * dot(N, I) + sqrt(k)) * N</code> , where <code>k = 1 - eta * eta * (1.0 - dot(N, I) * dot(N, I))</code> .	
<code>round</code>	Correctly rounded.	
<code>roundEven</code>	Correctly rounded.	
<code>trunc</code>	Correctly rounded.	

Instruction	Single precision, unless decorated with RelaxedPrecision	Half precision
<code>fabs</code>	Correct result.	
<code>fsign</code>	Correct result.	
<code>floor</code>	Correctly rounded.	
<code>ceil</code>	Correctly rounded.	
<code>fract</code>	Correctly rounded.	
<code>modf</code>	Correctly rounded.	
<code>fmin</code>	Correct result.	
<code>fmax</code>	Correct result.	
<code>fcamp</code>	Correct result.	
<code>fmix(x, y, a)</code>	Inherited from $x \times (1.0 - a) + y \times a$.	
<code>step</code>	Correctly rounded.	
<code>smoothStep(edge0, edge1, x)</code>	Inherited from $t \times t \times (3.0 - 2.0 \times t)$, where $t = clamp(\frac{x - edge0}{edge1 - edge0}, 0.0, 1.0)$.	
<code>nmin</code>	Correct result.	
<code>nmax</code>	Correct result.	
<code>ncamp</code>	Correct result.	
<code>packHalf2x16</code>	Correctly rounded with implementation defined rounding mode.	

GLSL.std.450 extended instructions specifically defined in terms of the above instructions inherit the above errors. GLSL.std.450 extended instructions not listed above and not defined in terms of the above have undefined precision.

If either operand to `OpSRem` and `OpSMod` instructions is negative the result is undefined.

NOTE While the `OpSRem` and `OpSMod` instructions are supported by the Vulkan environment, they require non-negative values and thus do not enable additional functionality beyond what `OpUMod` provides.

Buffer Indexing Calculations

Pipelines and shaders **may** be compiled to use 32-bit integer arithmetic to compute byte offsets (e.g. based on `OpAccessChain` instructions) within a physical storage buffer or buffer, limiting the accessible range to 4GB. For physical storage buffers, the base address is a 64-bit value and the implicit base plus offset calculation uses 64-bit addition.

NOTE Note that `OpAccessChain` indices are always treated as signed, so a 32-bit index can only address 2GB if the `ArrayStride` is 1.

The offset calculations that have the 4GB limit include:

- **OpAccessChain** and **OpPtrAccessChain** - the total offset summed over all indices multiplied by strides, including spanning multiple access chain instructions that lead to a given memory access.

Signedness of SPIR-V Image Accesses

SPIR-V associates a signedness with all integer image accesses. This is required in certain parts of the SPIR-V and the Vulkan image access pipeline to ensure defined results. The signedness is determined from a combination of the access instruction's **Image Operands** and the underlying image's **Sampled Type** as follows:

1. If the instruction's **Image Operands** contains the **SignExtend** operand then the access is signed.
2. If the instruction's **Image Operands** contains the **ZeroExtend** operand then the access is unsigned.
3. Otherwise, the image accesses signedness matches that of the **Sampled Type** of the **OpTypeImage** being accessed.

Image Format and Type Matching

When specifying the **Image Format** of an **OpTypeImage**, the converted bit width and type, as shown in the table below, **must** match the **Sampled Type**. The signedness **must** match the **signedness of any access** to the image.

NOTE

Formatted accesses are always converted from a shader readable type to the resource's format or vice versa via **Texel Decode** for reads and **Texel Encode** for writes. As such, the bit width and format below do not necessarily match 1:1 with what might be expected for some formats.

For a given **Image Format**, the **Sampled Type** **must** be the type described in the *Type* column of the below table, with its **Literal Width** set to that in the *Bit Width* column. Every access that is made to the image **must** have a signedness equal to that in the *Signedness* column (where applicable).

Image Format	Type-Declaration instructions	Bit Width	Signedness
Unknown	Any	Any	Any

Image Format	Type-Declaration instructions	Bit Width	Signedness
Rgba32f	OpTypeFloat	32	N/A
Rg32f			
R32f			
Rgba16f			
Rg16f			
R16f			
Rgba16			
Rg16			
R16			
Rgba16Snorm			
Rg16Snorm			
R16Snorm			
Rgb10A2			
R11fG11fB10f			
Rgba8			
Rg8			
R8			
Rgba8Snorm			
Rg8Snorm			
R8Snorm			

Image Format	Type-Declaration instructions	Bit Width	Signedness
Rgba32i	OpTypeInt	32	1
Rg32i			
R32i			
Rgba16i			
Rg16i			
R16i			
Rgba8i			
Rg8i			
R8i			
Rgba32ui			0
Rg32ui			
R32ui			
Rgba16ui			
Rg16ui			
R16ui			
Rgb10a2ui			
Rgba8ui			
Rg8ui			
R8ui			
R64i	OpTypeInt	64	1
R64ui			0

The *SPIR-V Type* is defined by an instruction in SPIR-V, declared with the Type-Declaration Instruction, Bit Width, and Signedness from above.

Compatibility Between SPIR-V Image Dimensions and Vulkan ImageView Types

SPIR-V **Image Dim** values are compatible with **VkImageView viewType** values as defined below:

Table 83. SPIR-V and Vulkan ImageView Dimension Compatibility

SPIR-V Image Dim	Compatible Vulkan ImageView viewType
1D	VK_IMAGE_VIEW_TYPE_1D, VK_IMAGE_VIEW_TYPE_1D_ARRAY
2D	VK_IMAGE_VIEW_TYPE_2D, VK_IMAGE_VIEW_TYPE_2D_ARRAY
3D	VK_IMAGE_VIEW_TYPE_3D
Cube	VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_CUBE_ARRAY

Compatibility Between SPIR-V Image Formats and Vulkan Formats

SPIR-V **Image Format** values are compatible with **VkFormat** values as defined below:

Table 84. SPIR-V and Vulkan Image Format Compatibility

SPIR-V Image Format	Compatible Vulkan Format
Unknown	Any
R8	VK_FORMAT_R8_UNORM
R8Snorm	VK_FORMAT_R8_SNORM
R8ui	VK_FORMAT_R8_UINT
R8i	VK_FORMAT_R8_SINT
Rg8	VK_FORMAT_R8G8_UNORM
Rg8Snorm	VK_FORMAT_R8G8_SNORM
Rg8ui	VK_FORMAT_R8G8_UINT
Rg8i	VK_FORMAT_R8G8_SINT
Rgba8	VK_FORMAT_R8G8B8A8_UNORM
Rgba8Snorm	VK_FORMAT_R8G8B8A8_SNORM
Rgba8ui	VK_FORMAT_R8G8B8A8_UINT
Rgba8i	VK_FORMAT_R8G8B8A8_SINT
Rgb10A2	VK_FORMAT_A2B10G10R10_UNORM_PACK32
Rgb10a2ui	VK_FORMAT_A2B10G10R10_UINT_PACK32
R16	VK_FORMAT_R16_UNORM
R16Snorm	VK_FORMAT_R16_SNORM
R16ui	VK_FORMAT_R16_UINT
R16i	VK_FORMAT_R16_SINT
R16f	VK_FORMAT_R16_SFLOAT
Rg16	VK_FORMAT_R16G16_UNORM
Rg16Snorm	VK_FORMAT_R16G16_SNORM
Rg16ui	VK_FORMAT_R16G16_UINT
Rg16i	VK_FORMAT_R16G16_SINT
Rg16f	VK_FORMAT_R16G16_SFLOAT
Rgba16	VK_FORMAT_R16G16B16A16_UNORM
Rgba16Snorm	VK_FORMAT_R16G16B16A16_SNORM
Rgba16ui	VK_FORMAT_R16G16B16A16_UINT
Rgba16i	VK_FORMAT_R16G16B16A16_SINT
Rgba16f	VK_FORMAT_R16G16B16A16_SFLOAT
R32ui	VK_FORMAT_R32_UINT
R32i	VK_FORMAT_R32_SINT
R32f	VK_FORMAT_R32_SFLOAT

SPIR-V Image Format	Compatible Vulkan Format
Rg32ui	VK_FORMAT_R32G32_UINT
Rg32i	VK_FORMAT_R32G32_SINT
Rg32f	VK_FORMAT_R32G32_SFLOAT
Rgba32ui	VK_FORMAT_R32G32B32A32_UINT
Rgba32i	VK_FORMAT_R32G32B32A32_SINT
Rgba32f	VK_FORMAT_R32G32B32A32_SFLOAT
R64ui	VK_FORMAT_R64_UINT
R64i	VK_FORMAT_R64_SINT
R11fG11fB10f	VK_FORMAT_B10G11R11_UFLOAT_PACK32

Appendix B: Memory Model

NOTE

This memory model describes synchronizations provided by all implementations; however, some of the synchronizations defined require extra features to be supported by the implementation. See [VkPhysicalDeviceVulkanMemoryModelFeatures](#).

Agent

Operation is a general term for any task that is executed on the system.

NOTE

An operation is by definition something that is executed. Thus if an instruction is skipped due to control flow, it does not constitute an operation.

Each operation is executed by a particular *agent*. Possible agents include each:

- fixed-function stage of the pipeline,
- shader invocation, or
- host thread.

Memory Location

A *memory location* identifies unique storage for 8 bits of data. Memory operations access a *set of memory locations* consisting of one or more memory locations at a time, e.g. an operation accessing a 32-bit integer in memory would read/write a set of four memory locations. Memory operations that access whole aggregates **may** access any padding bytes between elements or members, but no padding bytes at the end of the aggregate. Two sets of memory locations *overlap* if the intersection of their sets of memory locations is non-empty. A memory operation **must** not affect memory at a memory location not within its set of memory locations.

Memory locations for buffers and images are explicitly allocated in [VkDeviceMemory](#) objects , and are implicitly allocated for SPIR-V variables in each shader invocation .

Allocation

The values stored in newly allocated memory locations are determined by a SPIR-V variable's initializer, if present, or else are undefined. At the time an allocation is created there have been no [memory operations](#) to any of its memory locations. The initialization is not considered to be a memory operation.

NOTE

For tessellation control shader output variables, a consequence of initialization not being considered a memory operation is that some implementations may need to insert a barrier between the initialization of the output variables and any reads of those variables.

Memory Operation

For an operation A and memory location M:

- A *reads* M if and only if the data stored in M is an input to A.
- A *writes* M if and only if the data output from A is stored to M.
- A *accesses* M if and only if it either reads or writes (or both) M.

NOTE

A write whose value is the same as what was already in those memory locations is still considered to be a write and has all the same effects.

Reference

A *reference* is an object that a particular agent **can** use to access a set of memory locations. On the host, a reference is a host virtual address. On the device, a reference is:

- The descriptor that a variable is bound to, for variables in Image, Uniform, or StorageBuffer storage classes. If the variable is an array (or array of arrays, etc.) then each element of the array **may** be a unique reference.
- The address range for a buffer in **PhysicalStorageBuffer** storage class, where the base of the address range is queried with **vkGetBufferDeviceAddress** and the length of the range is the size of the buffer.
- The variable itself for variables in other storage classes.

Two memory accesses through distinct references **may** require availability and visibility operations as defined [below](#).

Program-Order

A *dynamic instance* of an instruction is defined in SPIR-V (<https://registry.khronos.org/spir-v/specs/unified1/SPIRV.html#DynamicInstance>) as a way of referring to a particular execution of a static instruction. Program-order is an ordering on dynamic instances of instructions executed by a single shader invocation:

- (Basic block): If instructions A and B are in the same basic block, and A is listed in the module before B, then the n'th dynamic instance of A is program-ordered before the n'th dynamic instance of B.
- (Branch): The dynamic instance of a branch or switch instruction is program-ordered before the dynamic instance of the OpLabel instruction to which it transfers control.
- (Call entry): The dynamic instance of an **OpFunctionCall** instruction is program-ordered before the dynamic instances of the **OpFunctionParameter** instructions and the body of the called function.
- (Call exit): The dynamic instance of the instruction following an **OpFunctionCall** instruction is program-ordered after the dynamic instance of the return instruction executed by the called function.

- (Transitive Closure): If dynamic instance A of any instruction is program-ordered before dynamic instance B of any instruction and B is program-ordered before dynamic instance C of any instruction then A is program-ordered before C.
- (Complete definition): No other dynamic instances are program-ordered.

For instructions executed on the host, the source language defines the program-order relation (e.g. as “sequenced-before”).

Scope

Atomic and barrier instructions include scopes which identify sets of shader invocations that **must** obey the requested ordering and atomicity rules of the operation, as defined below.

The various scopes are described in detail in [the Shaders chapter](#).

Atomic Operation

An *atomic operation* on the device is any SPIR-V operation whose name begins with **OpAtomic**. An atomic operation on the host is any operation performed with an `std::atomic` typed object.

Each atomic operation has a memory [scope](#) and a [semantics](#). Informally, the scope determines which other agents it is atomic with respect to, and the [semantics](#) constrains its ordering against other memory accesses.

Device atomic operations have explicit scopes and semantics. Each host atomic operation implicitly uses the **CrossDevice** scope, and uses a memory semantics equivalent to a C++ `std::memory_order` value of `relaxed`, `acquire`, `release`, `acq_rel`, or `seq_cst`.

Two atomic operations A and B are *potentially-mutually-ordered* if and only if all of the following are true:

- They access the same set of memory locations.
- They use the same reference.
- A is in the instance of B’s memory scope.
- B is in the instance of A’s memory scope.
- A and B are not the same operation (irreflexive).

Two atomic operations A and B are *mutually-ordered* if and only if they are potentially-mutually-ordered and any of the following are true:

- A and B are both device operations.
- A and B are both host operations.

NOTE

If two atomic operations are not mutually-ordered, and if their sets of memory locations overlap, then each **must** be synchronized against the other as if they were non-atomic operations.

Scoped Modification Order

For a given atomic write A, all atomic writes that are mutually-ordered with A occur in an order known as A's *scoped modification order*. A's scoped modification order relates no other operations.

NOTE

Invocations outside the instance of A's memory scope **may** observe the values at A's set of memory locations becoming visible to it in an order that disagrees with the scoped modification order.

NOTE

It is valid to have non-atomic operations or atomics in a different scope instance to the same set of memory locations, as long as they are synchronized against each other as if they were non-atomic (if they are not, it is treated as a [data race](#)). That means this definition of A's scoped modification order could include atomic operations that occur much later, after intervening non-atomics. That is a bit non-intuitive, but it helps to keep this definition simple and non-circular.

Memory Semantics

Non-atomic memory operations, by default, **may** be observed by one agent in a different order than they were written by another agent.

Atomics and some synchronization operations include *memory semantics*, which are flags that constrain the order in which other memory accesses (including non-atomic memory accesses and [availability and visibility operations](#)) performed by the same agent **can** be observed by other agents, or **can** observe accesses by other agents.

Host instructions that include semantics are some `std::atomic` methods and memory fences. Device instructions that include semantics are `OpAtomic*`, `OpControlBarrier`, `OpMemoryBarrier`, and `OpMemoryNamedBarrier`.

Vulkan supports the following memory semantics:

- Relaxed: No constraints on order of other memory accesses.
- Acquire: A memory read with this semantic performs an *acquire operation*. A memory barrier with this semantic is an *acquire barrier*.
- Release: A memory write with this semantic performs a *release operation*. A memory barrier with this semantic is a *release barrier*.
- AcquireRelease: A memory read-modify-write operation with this semantic performs both an acquire operation and a release operation, and inherits the limitations on ordering from both of those operations. A memory barrier with this semantic is both a release and acquire barrier.

SequentiallyConsistent memory semantics is not supported and **must** not be used.

NOTE

SPIR-V does not support “consume” semantics on the device.

The memory semantics operand **can** also include *storage class semantics* flags which indicate the storage classes constrained by the synchronization. Vulkan supports the following SPIR-V storage

class semantics flags:

- UniformMemory
- WorkgroupMemory
- ImageMemory
- OutputMemory

The remaining storage class semantics flags (SubgroupMemory, CrossWorkgroupMemory, and AtomicCounterMemory) are ignored.

Each SPIR-V memory operation accesses a single storage class. Semantics in synchronization operations can include a combination of storage classes.

The UniformMemory storage class semantic applies to accesses to memory in the PhysicalStorageBuffer, Uniform and StorageBuffer storage classes. The WorkgroupMemory storage class semantic applies to accesses to memory in the Workgroup storage class. The ImageMemory storage class semantic applies to accesses to memory in the Image storage class. The OutputMemory storage class semantic applies to accesses to memory in the Output storage class.

NOTE

Informally, these constraints limit how memory operations can be reordered, and these limits apply not only to the order of accesses as performed in the agent that executes the instruction, but also to the order the effects of writes become visible to all other agents within the same instance of the instruction's memory scope.

NOTE

Release and acquire operations in different threads **can** act as synchronization operations, to guarantee that writes that happened before the release are visible after the acquire. (This is not a formal definition, just an Informative forward reference.)

NOTE

The OutputMemory storage class semantic is only useful in tessellation control shaders, which is the only execution model where output variables are shared between invocations.

The memory semantics operand **can** also include availability and visibility flags, which apply availability and visibility operations as described in [availability and visibility](#). The availability/visibility flags are:

- MakeAvailable: Semantics **must** be Release or AcquireRelease. Performs an availability operation before the release operation or barrier.
- MakeVisible: Semantics **must** be Acquire or AcquireRelease. Performs a visibility operation after the acquire operation or barrier.

The specifics of these operations are defined in [Availability and Visibility Semantics](#).

Host atomic operations **may** support a different list of memory semantics and synchronization operations, depending on the host architecture and source language.

Release Sequence

After an atomic operation A performs a release operation on a set of memory locations M, the *release sequence headed by A* is the longest continuous subsequence of A's scoped modification order that consists of:

- the atomic operation A as its first element
- atomic read-modify-write operations on M by any agent

NOTE

The atomics in the last bullet **must** be mutually-ordered with A by virtue of being in A's scoped modification order.

NOTE

This intentionally omits “atomic writes to M performed by the same agent that performed A”, which is present in the corresponding C++ definition.

Synchronizes-With

Synchronizes-with is a relation between operations, where each operation is either an atomic operation or a memory barrier (aka fence on the host).

If A and B are atomic operations, then A synchronizes-with B if and only if all of the following are true:

- A performs a release operation
- B performs an acquire operation
- A and B are mutually-ordered
- B reads a value written by A or by an operation in the release sequence headed by A

`OpControlBarrier`, `OpMemoryBarrier`, and `OpMemoryNamedBarrier` are *memory barrier* instructions in SPIR-V.

If A is a release barrier and B is an atomic operation that performs an acquire operation, then A synchronizes-with B if and only if all of the following are true:

- there exists an atomic write X (with any memory semantics)
- A is program-ordered before X
- X and B are mutually-ordered
- B reads a value written by X or by an operation in the release sequence headed by X
 - If X is relaxed, it is still considered to head a hypothetical release sequence for this rule
- A and B are in the instance of each other's memory scopes
- X's storage class is in A's semantics.

If A is an atomic operation that performs a release operation and B is an acquire barrier, then A synchronizes-with B if and only if all of the following are true:

- there exists an atomic read X (with any memory semantics)
- X is program-ordered before B
- X and A are mutually-ordered
- X reads a value written by A or by an operation in the release sequence headed by A
- A and B are in the instance of each other's memory scopes
- X's storage class is in B's semantics.

If A is a release barrier and B is an acquire barrier, then A synchronizes-with B if all of the following are true:

- there exists an atomic write X (with any memory semantics)
- A is program-ordered before X
- there exists an atomic read Y (with any memory semantics)
- Y is program-ordered before B
- X and Y are mutually-ordered
- Y reads the value written by X or by an operation in the release sequence headed by X
 - If X is relaxed, it is still considered to head a hypothetical release sequence for this rule
- A and B are in the instance of each other's memory scopes
- X's and Y's storage class is in A's and B's semantics.
 - NOTE: X and Y **must** have the same storage class, because they are mutually ordered.

If A is a release barrier, B is an acquire barrier, and C is a control barrier (where A **can** equal C, and B **can** equal C), then A synchronizes-with B if all of the following are true:

- A is program-ordered before (or equals) C
- C is program-ordered before (or equals) B
- A and B are in the instance of each other's memory scopes
- A and B are in the instance of C's execution scope

NOTE

This is similar to the barrier-barrier synchronization above, but with a control barrier filling the role of the relaxed atomics.

Let F be an ordering of fragment shader invocations, such that invocation F_1 is ordered before invocation F_2 if and only if F_1 and F_2 overlap as described in [Fragment Shader Interlock](#) and F_1 executes the interlocked code before F_2 .

If A is an `OpEndInvocationInterlockEXT` instruction and B is an `OpBeginInvocationInterlockEXT` instruction, then A synchronizes-with B if the agent that executes A is ordered before the agent that executes B in F. A and B are both considered to have `FragmentInterlock` memory scope and semantics of UniformMemory and ImageMemory, and A is considered to have Release semantics and B is considered to have Acquire semantics.

NOTE

`OpBeginInvocationInterlockEXT` and `OpBeginInvocationInterlockEXT` do not perform implicit availability or visibility operations. Usually, shaders using fragment shader interlock will declare the relevant resources as `coherent` to get implicit `per-instruction availability and visibility operations`.

No other release and acquire barriers synchronize-with each other.

System-Synchronizes-With

System-synchronizes-with is a relation between arbitrary operations on the device or host. Certain operations system-synchronize-with each other, which informally means the first operation occurs before the second and that the synchronization is performed without using application-visible memory accesses.

If there is an `execution dependency` between two operations A and B, then the operation in the first synchronization scope system-synchronizes-with the operation in the second synchronization scope.

NOTE

This covers all Vulkan synchronization primitives, including device operations executing before a synchronization primitive is signaled, wait operations happening before subsequent device operations, signal operations happening before host operations that wait on them, and host operations happening before `vkQueueSubmit`. The list is spread throughout the synchronization chapter, and is not repeated here.

System-synchronizes-with implicitly includes all storage class semantics and has `CrossDevice` scope.

If A system-synchronizes-with B, we also say A is *system-synchronized-before* B and B is *system-synchronized-after* A.

Private vs. Non-Private

By default, non-atomic memory operations are treated as *private*, meaning such a memory operation is not intended to be used for communication with other agents.

Non-private memory operations are intended to be used for communication with other agents. Atomic operations are always considered *non-private*.

Memory operations with the `NonPrivatePointer`, `NonPrivateTexel` bit set are treated as *non-private*

More precisely, for private memory operations to be `Location-Ordered` between distinct agents requires using system-synchronizes-with rather than shader-based synchronization. Private memory operations still obey program-order.

Inter-Thread-Happens-Before

Let SC be a non-empty set of storage class semantics. Then (using template syntax) operation A *inter-thread-happens-before*<SC> operation B if and only if any of the following is true:

- A system-synchronizes-with B
- A synchronizes-with B, and both A and B have all of SC in their semantics
- A is an operation on memory in a storage class in SC or that has all of SC in its semantics, B is a release barrier or release atomic with all of SC in its semantics, and A is program-ordered before B
- A is an acquire barrier or acquire atomic with all of SC in its semantics, B is an operation on memory in a storage class in SC or that has all of SC in its semantics, and A is program-ordered before B
- A and B are both host operations and A inter-thread-happens-before B as defined in the host language specification
- A inter-thread-happens-before<SC> some X and X inter-thread-happens-before<SC> B

Happens-Before

Operation A *happens-before* operation B if and only if any of the following is true:

- A is program-ordered before B
- A inter-thread-happens-before<SC> B for some set of storage classes SC

Happens-after is defined similarly.

NOTE

Unlike C++, happens-before is not always sufficient for a write to be visible to a read. Additional [availability and visibility](#) operations **may** be required for writes to be [visible-to](#) other memory accesses.

NOTE

Happens-before is not transitive, but each of program-order and inter-thread-happens-before<SC> are transitive. These can be thought of as covering the “single-threaded” case and the “multi-threaded” case, and it is not necessary (and not valid) to form chains between the two.

Availability and Visibility

Availability and *visibility* are states of a write operation, which (informally) track how far the write has permeated the system, i.e. which agents and references are able to observe the write. Availability state is per *memory domain*. Visibility state is per (agent,reference) pair. Availability and visibility states are per-memory location for each write.

Memory domains are named according to the agents whose memory accesses use the domain. Domains used by shader invocations are organized hierarchically into multiple smaller memory domains which correspond to the different [scopes](#). Each memory domain is considered the *dual* of a scope, and vice versa. The memory domains defined in Vulkan include:

- *host* - accessible by host agents
- *device* - accessible by all device agents for a particular device

- *shader* - accessible by shader agents for a particular device, corresponding to the **Device** scope
- *queue family instance* - accessible by shader agents in a single queue family, corresponding to the **QueueFamily** scope.
- *fragment interlock instance* - accessible by fragment shader agents that **overlap**, corresponding to the **FragmentInterlock** scope.
- *workgroup instance* - accessible by shader agents in the same workgroup, corresponding to the **Workgroup** scope.
- *subgroup instance* - accessible by shader agents in the same subgroup, corresponding to the **Subgroup** scope.

The memory domains are nested in the order listed above, with memory domains later in the list nested in the domains earlier in the list.

NOTE

Memory domains do not correspond to storage classes or device-local and host-local **VkDeviceMemory** allocations, rather they indicate whether a write can be made visible only to agents in the same subgroup, same workgroup, in overlapping fragment shader invocation, in any shader invocation, or anywhere on the device, or host. The shader, queue family instance, fragment interlock instance, workgroup instance, and subgroup instance domains are only used for shader-based availability/visibility operations, in other cases writes can be made available from/visible to the shader via the device domain.

Availability operations, *visibility operations*, and *memory domain operations* alter the state of the write operations that happen-before them, and which are included in their *source scope* to be available or visible to their *destination scope*.

- For an availability operation, the source scope is a set of (agent,reference,memory location) tuples, and the destination scope is a set of memory domains.
- For a memory domain operation, the source scope is a memory domain and the destination scope is a memory domain.
- For a visibility operation, the source scope is a set of memory domains and the destination scope is a set of (agent,reference,memory location) tuples.

How the scopes are determined depends on the specific operation. Availability and memory domain operations expand the set of memory domains to which the write is available. Visibility operations expand the set of (agent,reference,memory location) tuples to which the write is visible.

Recall that availability and visibility states are per-memory location, and let W be a write operation to one or more locations performed by agent A via reference R. Let L be one of the locations written. (W,L) (the write W to L), is initially not available to any memory domain and only visible to (A,R,L). An availability operation AV that happens-after W and that includes (A,R,L) in its source scope makes (W,L) *available* to the memory domains in its destination scope.

A memory domain operation DOM that happens-after AV and for which (W,L) is available in the source scope makes (W,L) available in the destination memory domain.

A visibility operation VIS that happens-after AV (or DOM) and for which (W,L) is available in any

domain in the source scope makes (W,L) *visible* to all (agent,reference,L) tuples included in its destination scope.

If write W_2 happens-after W , and their sets of memory locations overlap, then W will not be available/visible to all agents/references for those memory locations that overlap (and future AV/DOM/VIS ops cannot revive W 's write to those locations).

Availability, memory domain, and visibility operations are treated like other non-atomic memory accesses for the purpose of [memory semantics](#), meaning they can be ordered by release-acquire sequences or memory barriers.

An *availability chain* is a sequence of availability operations to increasingly broad memory domains, where element $N+1$ of the chain is performed in the dual scope instance of the destination memory domain of element N and element N happens-before element $N+1$. An example is an availability operation with destination scope of the workgroup instance domain that happens-before an availability operation to the shader domain performed by an invocation in the same workgroup. An availability chain AVC that happens-after W and that includes (A,R,L) in the source scope makes (W,L) *available* to the memory domains in its final destination scope. An availability chain with a single element is just the availability operation.

Similarly, a *visibility chain* is a sequence of visibility operations from increasingly narrow memory domains, where element N of the chain is performed in the dual scope instance of the source memory domain of element $N+1$ and element N happens-before element $N+1$. An example is a visibility operation with source scope of the shader domain that happens-before a visibility operation with source scope of the workgroup instance domain performed by an invocation in the same workgroup. A visibility chain VISC that happens-after AVC (or DOM) and for which (W,L) is available in any domain in the source scope makes (W,L) *visible* to all (agent,reference,L) tuples included in its final destination scope. A visibility chain with a single element is just the visibility operation.

Availability, Visibility, and Domain Operations

The following operations generate availability, visibility, and domain operations. When multiple availability/visibility/domain operations are described, they are system-synchronized-with each other in the order listed.

An operation that performs a [memory dependency](#) generates:

- If the source access mask includes `VK_ACCESS_HOST_WRITE_BIT`, then the dependency includes a memory domain operation from host domain to device domain.
- An availability operation with source scope of all writes in the first [access scope](#) of the dependency and a destination scope of the device domain.
- A visibility operation with source scope of the device domain and destination scope of the second access scope of the dependency.
- If the destination access mask includes `VK_ACCESS_HOST_READ_BIT` or `VK_ACCESS_HOST_WRITE_BIT`, then the dependency includes a memory domain operation from device domain to host domain.

[vkFlushMappedMemoryRanges](#) performs an availability operation, with a source scope of

(agents,references) = (all host threads, all mapped memory ranges passed to the command), and destination scope of the host domain.

`vkInvalidateMappedMemoryRanges` performs a visibility operation, with a source scope of the host domain and a destination scope of (agents,references) = (all host threads, all mapped memory ranges passed to the command).

`vkQueueSubmit` performs a memory domain operation from host to device, and a visibility operation with source scope of the device domain and destination scope of all agents and references on the device.

Availability and Visibility Semantics

A memory barrier or atomic operation via agent A that includes `MakeAvailable` in its semantics performs an availability operation whose source scope includes agent A and all references in the storage classes in that instruction's storage class semantics, and all memory locations, and whose destination scope is a set of memory domains selected as specified below. The implicit availability operation is program-ordered between the barrier or atomic and all other operations program-ordered before the barrier or atomic.

A memory barrier or atomic operation via agent A that includes `MakeVisible` in its semantics performs a visibility operation whose source scope is a set of memory domains selected as specified below, and whose destination scope includes agent A and all references in the storage classes in that instruction's storage class semantics, and all memory locations. The implicit visibility operation is program-ordered between the barrier or atomic and all other operations program-ordered after the barrier or atomic.

The memory domains are selected based on the memory scope of the instruction as follows:

- `Device` scope uses the shader domain
- `QueueFamily` scope uses the queue family instance domain
- `FragmentInterlock` scope uses the fragment interlock instance domain
- `Workgroup` scope uses the workgroup instance domain
- `Subgroup` uses the subgroup instance domain
- `Invocation` perform no availability/visibility operations.

When an availability operation performed by an agent A includes a memory domain D in its destination scope, where D corresponds to scope instance S, it also includes the memory domains that correspond to each smaller scope instance S' that is a subset of S and that includes A. Similarly for visibility operations.

Per-Instruction Availability and Visibility Semantics

A memory write instruction that includes `MakePointerAvailable`, or an image write instruction that includes `MakeTexelAvailable`, performs an availability operation whose source scope includes the agent and reference used to perform the write and the memory locations written by the

instruction, and whose destination scope is a set of memory domains selected by the Scope operand specified in [Availability and Visibility Semantics](#). The implicit availability operation is program-ordered between the write and all other operations program-ordered after the write.

A memory read instruction that includes `MakePointerVisible`, or an image read instruction that includes `MakeTexelVisible`, performs a visibility operation whose source scope is a set of memory domains selected by the Scope operand as specified in [Availability and Visibility Semantics](#), and whose destination scope includes the agent and reference used to perform the read and the memory locations read by the instruction. The implicit visibility operation is program-ordered between read and all other operations program-ordered before the read.

NOTE

Although reads with per-instruction visibility only perform visibility ops from the shader or fragment interlock instance or workgroup instance or subgroup instance domain, they will also see writes that were made visible via the device domain, i.e. those writes previously performed by non-shader agents and made visible via API commands.

NOTE

It is expected that all invocations in a subgroup execute on the same processor with the same path to memory, and thus availability and visibility operations with subgroup scope can be expected to be “free”.

Location-Ordered

Let X and Y be memory accesses to overlapping sets of memory locations M , where $X \neq Y$. Let (A_x, R_x) be the agent and reference used for X , and (A_y, R_y) be the agent and reference used for Y . For now, let “ $\rightarrow$ ” denote happens-before and “ $\rightarrow^{rcpo}$ ” denote the reflexive closure of program-ordered before.

If D_1 and D_2 are different memory domains, then let $DOM(D_1, D_2)$ be a memory domain operation from D_1 to D_2 . Otherwise, let $DOM(D, D)$ be a placeholder such that $X \rightarrow DOM(D, D) \rightarrow Y$ if and only if $X \rightarrow Y$.

X is *location-ordered* before Y for a location L in M if and only if any of the following is true:

- $A_x == A_y$ and $R_x == R_y$ and $X \rightarrow Y$
 - NOTE: this case means no availability/visibility ops are required when it is the same (agent,reference).
- X is a read, both X and Y are non-private, and $X \rightarrow Y$
- X is a read, and X (transitively) system-synchronizes with Y
- If $R_x == R_y$ and A_x and A_y access a common memory domain D (e.g. are in the same workgroup instance if D is the workgroup instance domain), and both X and Y are non-private:
 - X is a write, Y is a write, $AVC(A_x, R_x, D, L)$ is an availability chain making (X, L) available to domain D , and $X \rightarrow^{rcpo} AVC(A_x, R_x, D, L) \rightarrow Y$
 - X is a write, Y is a read, $AVC(A_x, R_x, D, L)$ is an availability chain making (X, L) available to domain D , $VISC(A_y, R_y, D, L)$ is a visibility chain making writes to L available in domain D visible to Y , and $X \rightarrow^{rcpo} AVC(A_x, R_x, D, L) \rightarrow VISC(A_y, R_y, D, L) \rightarrow^{rcpo} Y$

- If `VkPhysicalDeviceVulkanMemoryModelFeatures::vulkanMemoryModelAvailabilityVisibilityChains` is `VK_FALSE`, then AVC and VISC **must** each only have a single element in the chain, in each sub-bullet above.
- Let D_x and D_y each be either the device domain or the host domain, depending on whether A_x and A_y execute on the device or host:
 - X is a write and Y is a write, and $X \rightarrow AV(A_x, R_x, D_x, L) \rightarrow DOM(D_x, D_y) \rightarrow Y$
 - X is a write and Y is a read, and $X \rightarrow AV(A_x, R_x, D_x, L) \rightarrow DOM(D_x, D_y) \rightarrow VIS(A_y, R_y, D_y, L) \rightarrow Y$

NOTE

The final bullet (synchronization through device/host domain) requires API-level synchronization operations, since the device/host domains are not accessible via shader instructions. And “device domain” is not to be confused with “device scope”, which synchronizes through the “shader domain”.

Data Race

Let X and Y be operations that access overlapping sets of memory locations M, where $X \neq Y$, and at least one of X and Y is a write, and X and Y are not mutually-ordered atomic operations. If there does not exist a location-ordered relation between X and Y for each location in M, then there is a *data race*.

Applications **must** ensure that no data races occur during the execution of their application.

NOTE

Data races can only occur due to instructions that are actually executed. For example, an instruction skipped due to control flow must not contribute to a data race.

Visible-To

Let X be a write and Y be a read whose sets of memory locations overlap, and let M be the set of memory locations that overlap. Let M_2 be a non-empty subset of M. Then X is *visible-to* Y for memory locations M_2 if and only if all of the following are true:

- X is location-ordered before Y for each location L in M_2 .
- There does not exist another write Z to any location L in M_2 such that X is location-ordered before Z for location L and Z is location-ordered before Y for location L.

If X is visible-to Y, then Y reads the value written by X for locations M_2 .

NOTE

It is possible for there to be a write between X and Y that overwrites a subset of the memory locations, but the remaining memory locations (M_2) will still be visible-to Y.

Acyclicity

Reads-from is a relation between operations, where the first operation is a write, the second operation is a read, and the second operation reads the value written by the first operation. *From-*

reads is a relation between operations, where the first operation is a read, the second operation is a write, and the first operation reads a value written earlier than the second operation in the second operation's scoped modification order or location order (or the first operation reads from the initial value, and the second operation is any write to the same locations).

Then the implementation **must** guarantee that no cycles exist in the union of the following relations:

- location-ordered
- scoped modification order (over all atomic writes)
- reads-from
- from-reads

NOTE

This is a “consistency” axiom, which informally guarantees that sequences of operations cannot violate causality.

Scoped Modification Order Coherence

Let A and B be mutually-ordered atomic operations, where A is location-ordered before B. Then the following rules are a consequence of acyclicity:

- If A and B are both reads and A does not read the initial value, then the write that A takes its value from **must** be earlier in its own scoped modification order than (or the same as) the write that B takes its value from (no cycles between location-order, reads-from, and from-reads).
- If A is a read and B is a write and A does not read the initial value, then A **must** take its value from a write earlier than B in B's scoped modification order (no cycles between location-order, scope modification order, and reads-from).
- If A is a write and B is a read, then B **must** take its value from A or a write later than A in A's scoped modification order (no cycles between location-order, scoped modification order, and from-reads).
- If A and B are both writes, then A **must** be earlier than B in A's scoped modification order (no cycles between location-order and scoped modification order).
- If A is a write and B is a read-modify-write and B reads the value written by A, then B comes immediately after A in A's scoped modification order (no cycles between scoped modification order and from-reads).

Shader I/O

If a shader invocation A in a shader stage other than **Vertex** performs a memory read operation X from an object in storage class **Input**, then X is system-synchronized-after all writes to the corresponding **Output** storage variable(s) in the shader invocation(s) that contribute to generating invocation A, and those writes are all visible-to X.

NOTE

It is not necessary for the upstream shader invocations to have completed execution, they only need to have generated the output that is being read.

Deallocation

The deallocation of SPIR-V variables is managed by the system and happens-after all operations on those variables.

Descriptions (Informative)

This subsection offers more easily understandable consequences of the memory model for app/compiler developers.

Let SC be the storage class(es) specified by a release or acquire operation or barrier.

- An atomic write with release semantics must not be reordered against any read or write to SC that is program-ordered before it (regardless of the storage class the atomic is in).
- An atomic read with acquire semantics must not be reordered against any read or write to SC that is program-ordered after it (regardless of the storage class the atomic is in).
- Any write to SC program-ordered after a release barrier must not be reordered against any read or write to SC program-ordered before that barrier.
- Any read from SC program-ordered before an acquire barrier must not be reordered against any read or write to SC program-ordered after the barrier.

A control barrier (even if it has no memory semantics) must not be reordered against any memory barriers.

This memory model allows memory accesses with and without availability and visibility operations, as well as atomic operations, all to be performed on the same memory location. This is critical to allow it to reason about memory that is reused in multiple ways, e.g. across the lifetime of different shader invocations or draw calls. While GLSL (and legacy SPIR-V) applies the “coherent” decoration to variables (for historical reasons), this model treats each memory access instruction as having optional implicit availability/visibility operations. GLSL to SPIR-V compilers should map all (non-atomic) operations on a coherent variable to `Make{Pointer, Texel}{Available}{Visible}` flags in this model.

Atomic operations implicitly have availability/visibility operations, and the scope of those operations is taken from the atomic operation’s scope.

Tessellation Output Ordering

For SPIR-V that uses the Vulkan Memory Model, the `OutputMemory` storage class is used to synchronize accesses to tessellation control output variables. For legacy SPIR-V that does not enable the Vulkan Memory Model via `OpMemoryModel`, tessellation outputs can be ordered using a control barrier with no particular memory scope or semantics, as defined below.

Let X and Y be memory operations performed by shader invocations A_x and A_y . Operation X is *tessellation-output-ordered* before operation Y if and only if all of the following are true:

- There is a dynamic instance of an `OpControlBarrier` instruction C such that X is program-ordered

before C in A_x and C is program-ordered before Y in A_y .

- A_x and A_y are in the same instance of C's execution scope.

If shader invocations A_x and A_y in the **TessellationControl** execution model execute memory operations X and Y, respectively, on the **Output** storage class, and X is tessellation-output-ordered before Y with a scope of **Workgroup**, then X is location-ordered before Y, and if X is a write and Y is a read then X is visible-to Y.

Appendix C: Compressed Image Formats

The compressed texture formats used by Vulkan are described in the specifically identified sections of the [Khronos Data Format Specification](#), version 1.3.

Unless otherwise described, the quantities encoded in these compressed formats are treated as normalized, unsigned values.

Those formats listed as sRGB-encoded have in-memory representations of R, G and B components which are nonlinearly-encoded as R', G', and B'; any alpha component is unchanged. As part of filtering, the nonlinear R', G', and B' values are converted to linear R, G, and B components; any alpha component is unchanged. The conversion between linear and nonlinear encoding is performed as described in the “KHR_DF_TRANSFER_SRGB” section of the Khronos Data Format Specification.

Block-Compressed Image Formats

BC1, BC2 and BC3 formats are described in “S3TC Compressed Texture Image Formats” chapter of the [Khronos Data Format Specification](#). BC4 and BC5 are described in the “RGTC Compressed Texture Image Formats” chapter. BC6H and BC7 are described in the “BPTC Compressed Texture Image Formats” chapter.

Table 85. Mapping of Vulkan BC Formats to Descriptions

VkFormat	Khronos Data Format Specification description
Formats described in the “S3TC Compressed Texture Image Formats” chapter	
VK_FORMAT_BC1_RGB_UNORM_BLOCK	BC1 with no alpha
VK_FORMAT_BC1_RGB_SRGB_BLOCK	BC1 with no alpha, sRGB-encoded
VK_FORMAT_BC1_RGBA_UNORM_BLOCK	BC1 with alpha
VK_FORMAT_BC1_RGBA_SRGB_BLOCK	BC1 with alpha, sRGB-encoded
VK_FORMAT_BC2_UNORM_BLOCK	BC2
VK_FORMAT_BC2_SRGB_BLOCK	BC2, sRGB-encoded
VK_FORMAT_BC3_UNORM_BLOCK	BC3
VK_FORMAT_BC3_SRGB_BLOCK	BC3, sRGB-encoded
Formats described in the “RGTC Compressed Texture Image Formats” chapter	
VK_FORMAT_BC4_UNORM_BLOCK	BC4 unsigned
VK_FORMAT_BC4_SNORM_BLOCK	BC4 signed
VK_FORMAT_BC5_UNORM_BLOCK	BC5 unsigned
VK_FORMAT_BC5_SNORM_BLOCK	BC5 signed
Formats described in the “BPTC Compressed Texture Image Formats” chapter	
VK_FORMAT_BC6H_UFLOAT_BLOCK	BC6H (unsigned version)
VK_FORMAT_BC6H_SFLOAT_BLOCK	BC6H (signed version)
VK_FORMAT_BC7_UNORM_BLOCK	BC7
VK_FORMAT_BC7_SRGB_BLOCK	BC7, sRGB-encoded

ETC Compressed Image Formats

The following formats are described in the “ETC2 Compressed Texture Image Formats” chapter of the [Khronos Data Format Specification](#).

Table 86. Mapping of Vulkan ETC Formats to Descriptions

VkFormat	Khronos Data Format Specification description
VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK	RGB ETC2
VK_FORMAT_ETC2_R8G8B8_SRGB_BLOCK	RGB ETC2 with sRGB encoding
VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK	RGB ETC2 with punch-through alpha
VK_FORMAT_ETC2_R8G8B8A1_SRGB_BLOCK	RGB ETC2 with punch-through alpha and sRGB
VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK	RGBA ETC2
VK_FORMAT_ETC2_R8G8B8A8_SRGB_BLOCK	RGBA ETC2 with sRGB encoding
VK_FORMAT_EAC_R11_UNORM_BLOCK	Unsigned R11 EAC
VK_FORMAT_EAC_R11_SNORM_BLOCK	Signed R11 EAC
VK_FORMAT_EAC_R11G11_UNORM_BLOCK	Unsigned RG11 EAC
VK_FORMAT_EAC_R11G11_SNORM_BLOCK	Signed RG11 EAC

ASTC Compressed Image Formats

ASTC formats are described in the “ASTC Compressed Texture Image Formats” chapter of the [Khronos Data Format Specification](#).

Table 87. Mapping of Vulkan ASTC Formats to Descriptions

VkFormat	Compressed texel block dimensions	Requested mode
VK_FORMAT_ASTC_4x4_UNORM_BLOCK	4 × 4	Linear LDR
VK_FORMAT_ASTC_4x4_SRGB_BLOCK	4 × 4	sRGB
VK_FORMAT_ASTC_5x4_UNORM_BLOCK	5 × 4	Linear LDR
VK_FORMAT_ASTC_5x4_SRGB_BLOCK	5 × 4	sRGB
VK_FORMAT_ASTC_5x5_UNORM_BLOCK	5 × 5	Linear LDR
VK_FORMAT_ASTC_5x5_SRGB_BLOCK	5 × 5	sRGB
VK_FORMAT_ASTC_6x5_UNORM_BLOCK	6 × 5	Linear LDR
VK_FORMAT_ASTC_6x5_SRGB_BLOCK	6 × 5	sRGB
VK_FORMAT_ASTC_6x6_UNORM_BLOCK	6 × 6	Linear LDR
VK_FORMAT_ASTC_6x6_SRGB_BLOCK	6 × 6	sRGB
VK_FORMAT_ASTC_8x5_UNORM_BLOCK	8 × 5	Linear LDR
VK_FORMAT_ASTC_8x5_SRGB_BLOCK	8 × 5	sRGB
VK_FORMAT_ASTC_8x6_UNORM_BLOCK	8 × 6	Linear LDR
VK_FORMAT_ASTC_8x6_SRGB_BLOCK	8 × 6	sRGB
VK_FORMAT_ASTC_8x8_UNORM_BLOCK	8 × 8	Linear LDR
VK_FORMAT_ASTC_8x8_SRGB_BLOCK	8 × 8	sRGB
VK_FORMAT_ASTC_10x5_UNORM_BLOCK	10 × 5	Linear LDR
VK_FORMAT_ASTC_10x5_SRGB_BLOCK	10 × 5	sRGB
VK_FORMAT_ASTC_10x6_UNORM_BLOCK	10 × 6	Linear LDR
VK_FORMAT_ASTC_10x6_SRGB_BLOCK	10 × 6	sRGB
VK_FORMAT_ASTC_10x8_UNORM_BLOCK	10 × 8	Linear LDR
VK_FORMAT_ASTC_10x8_SRGB_BLOCK	10 × 8	sRGB
VK_FORMAT_ASTC_10x10_UNORM_BLOCK	10 × 10	Linear LDR
VK_FORMAT_ASTC_10x10_SRGB_BLOCK	10 × 10	sRGB
VK_FORMAT_ASTC_12x10_UNORM_BLOCK	12 × 10	Linear LDR
VK_FORMAT_ASTC_12x10_SRGB_BLOCK	12 × 10	sRGB
VK_FORMAT_ASTC_12x12_UNORM_BLOCK	12 × 12	Linear LDR

VkFormat	Compressed texel block dimensions	Requested mode
VK_FORMAT_ASTC_12x12_SRGB_BLOCK	12 × 12	sRGB
VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK	4 × 4	HDR
VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK	5 × 4	HDR
VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK	5 × 5	HDR
VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK	6 × 5	HDR
VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK	6 × 6	HDR
VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK	8 × 5	HDR
VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK	8 × 6	HDR
VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK	8 × 8	HDR
VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK	10 × 5	HDR
VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK	10 × 6	HDR
VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK	10 × 8	HDR
VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK	10 × 10	HDR
VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK	12 × 10	HDR
VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK	12 × 12	HDR

ASTC textures containing HDR block encodings **should** be passed to the API using an ASTC SFLOAT texture format.

NOTE

An HDR block in a texture passed using a LDR UNORM format will return the appropriate ASTC error color if the implementation supports only the ASTC LDR profile, but may result in either the error color or a decompressed HDR color if the implementation supports HDR decoding.

ASTC Decode Mode

If the `VK_EXT_astc_decode_mode` extension is enabled, the decode mode is determined as follows:

Table 88. Mapping of Vulkan ASTC Decoding Format to ASTC Decoding Modes

VkFormat	Decoding mode
VK_FORMAT_R16G16B16A16_SFLOAT	decode_float16
VK_FORMAT_R8G8B8A8_UNORM	decode_unorm8
VK_FORMAT_E5B9G9R9_UFLOAT_PACK32	decode_rgb9e5

Otherwise, the ASTC decode mode is `decode_float16`.

Note that an implementation **may** use HDR mode when linear LDR mode is requested unless the decode mode is `decode_unorm8`.

Appendix D: Core Revisions (Informative)

New minor versions of the Vulkan API are defined periodically by the Khronos Vulkan Working Group. These consist of some amount of additional functionality added to the core API, potentially including both new functionality and functionality [promoted](#) from extensions.

It is possible to build the specification for earlier versions, but to aid readability of the latest versions, this appendix gives an overview of the changes as compared to earlier versions.

Vulkan Version 1.2

Vulkan Version 1.2 [promoted](#) a number of key extensions into the core API:

- `VK_KHR_8bit_storage`
- `VK_KHR_buffer_device_address`
- `VK_KHR_create_renderpass2`
- `VK_KHR_depth_stencil_resolve`
- `VK_KHR_draw_indirect_count`
- `VK_KHR_driver_properties`
- `VK_KHR_image_format_list`
- `VK_KHR_imageless_framebuffer`
- `VK_KHR_sampler_mirror_clamp_to_edge`
- `VK_KHR_separate_depth_stencil_layouts`
- `VK_KHR_shader_atomic_int64`
- `VK_KHR_shader_float16_int8`
- `VK_KHR_shader_float_controls`
- `VK_KHR_shader_subgroup_extended_types`
- `VK_KHR_spirv_1_4`
- `VK_KHR_timeline_semaphore`
- `VK_KHR_uniform_buffer_standard_layout`
- `VK_KHR_vulkan_memory_model`
- `VK_EXT_descriptor_indexing`
- `VK_EXT_host_query_reset`
- `VK_EXT_sampler_filter_minmax`
- `VK_EXT_scalar_block_layout`
- `VK_EXT_separate_stencil_usage`
- `VK_EXT_shader_viewport_index_layer`

All differences in behavior between these extensions and the corresponding Vulkan 1.2

functionality are summarized below.

Differences Relative to `VK_KHR_8bit_storage`

If the `VK_KHR_8bit_storage` extension is not supported, support for the SPIR-V `storageBuffer8BitAccess` capability in shader modules is optional. Support for this feature is defined by `VkPhysicalDeviceVulkan12Features::storageBuffer8BitAccess` when queried via `vkGetPhysicalDeviceFeatures2`.

Differences Relative to `VK_KHR_draw_indirect_count`

If the `VK_KHR_draw_indirect_count` extension is not supported, support for the commands `vkCmdDrawIndirectCount` and `vkCmdDrawIndexedIndirectCount` is optional. Support for this feature is defined by `VkPhysicalDeviceVulkan12Features::drawIndirectCount` when queried via `vkGetPhysicalDeviceFeatures2`.

Differences Relative to `VK_KHR_sampler_mirror_clamp_to_edge`

If the `VK_KHR_sampler_mirror_clamp_to_edge` extension is not supported, support for the `VkSamplerAddressMode` `VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE` is optional. Support for this feature is defined by `VkPhysicalDeviceVulkan12Features::samplerMirrorClampToEdge` when queried via `vkGetPhysicalDeviceFeatures2`.

Differences Relative to `VK_EXT_descriptor_indexing`

If the `VK_EXT_descriptor_indexing` extension is not supported, support for the `descriptorIndexing` feature is optional. Support for this feature is defined by `VkPhysicalDeviceVulkan12Features::descriptorIndexing` when queried via `vkGetPhysicalDeviceFeatures2`.

Differences Relative to `VK_EXT_scalar_block_layout`

If the `VK_EXT_scalar_block_layout` extension is not supported, support for the `scalarBlockLayout` feature is optional. Support for this feature is defined by `VkPhysicalDeviceVulkan12Features::scalarBlockLayout` when queried via `vkGetPhysicalDeviceFeatures2`.

Differences Relative to `VK_EXT_shader_viewport_index_layer`

The `ShaderViewportIndexLayerEXT` SPIR-V capability was replaced with the `ShaderViewportIndex` and `ShaderLayer` capabilities. Declaring both is equivalent to declaring `ShaderViewportIndexLayerEXT`. If the `VK_EXT_shader_viewport_index_layer` extension is not supported, support for the `ShaderViewportIndexLayerEXT` SPIR-V capability is optional. Support for this feature is defined by `VkPhysicalDeviceVulkan12Features::shaderOutputViewportIndex` and `VkPhysicalDeviceVulkan12Features::shaderOutputLayer` when queried via `vkGetPhysicalDeviceFeatures2`.

Differences Relative to `VK_KHR_buffer_device_address`

If the `VK_KHR_buffer_device_address` extension is not supported, support for the `bufferDeviceAddress` feature is optional. Support for this feature is defined by `VkPhysicalDeviceVulkan12Features::bufferDeviceAddress` when queried via `vkGetPhysicalDeviceFeatures2`.

Differences Relative to `VK_KHR_shader_atomic_int64`

If the `VK_KHR_shader_atomic_int64` extension is not supported, support for the `shaderBufferInt64Atomics` feature is optional. Support for this feature is defined by

`VkPhysicalDeviceVulkan12Features::shaderBufferInt64Atomics` when queried via `vkGetPhysicalDeviceFeatures2`.

Differences Relative to `VK_KHR_shader_float16_int8`

If the `VK_KHR_shader_float16_int8` extension is not supported, support for the `shaderFloat16` and `shaderInt8` features is optional. Support for these features are defined by `VkPhysicalDeviceVulkan12Features::shaderFloat16` and `VkPhysicalDeviceVulkan12Features::shaderInt8` when queried via `vkGetPhysicalDeviceFeatures2`.

Differences Relative to `VK_KHR_vulkan_memory_model`

If the `VK_KHR_vulkan_memory_model` extension is not supported, support for the `vulkanMemoryModel` feature is optional. Support for this feature is defined by `VkPhysicalDeviceVulkan12Features::vulkanMemoryModel` when queried via `vkGetPhysicalDeviceFeatures2`.

Additional Vulkan 1.2 Feature Support

In addition to the promoted extensions described above, Vulkan 1.2 added support for:

- SPIR-V version 1.4.
- SPIR-V version 1.5.
- The `samplerMirrorClampToEdge` feature which indicates whether the implementation supports the `VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE` sampler address mode.
- The `ShaderNonUniform` capability in SPIR-V version 1.5.
- The `shaderOutputViewportIndex` feature which indicates that the `ShaderViewportIndex` capability can be used.
- The `shaderOutputLayer` feature which indicates that the `ShaderLayer` capability can be used.
- The `subgroupBroadcastDynamicId` feature which allows the “Id” operand of `OpGroupNonUniformBroadcast` to be dynamically uniform within a subgroup, and the “Index” operand of `OpGroupNonUniformQuadBroadcast` to be dynamically uniform within a derivative group, in shader modules of version 1.5 or higher.
- The `drawIndirectCount` feature which indicates whether the `vkCmdDrawIndirectCount` and `vkCmdDrawIndexedIndirectCount` functions can be used.
- The `descriptorIndexing` feature which indicates the implementation supports the minimum number of descriptor indexing features as defined in the [Feature Requirements](#) section.
- The `samplerFilterMinmax` feature which indicates whether the implementation supports the minimum number of image formats that support the `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT` feature bit as defined by the `filterMinmaxSingleComponentFormats` property minimum requirements.
- The `framebufferIntegerColorSampleCounts` limit which indicates the color sample counts that are supported for all framebuffer color attachments with integer formats.

New Macros

- `VK_API_VERSION_1_2`

New Commands

- [vkCmdBeginRenderPass2](#)
- [vkCmdDrawIndexedIndirectCount](#)
- [vkCmdDrawIndirectCount](#)
- [vkCmdEndRenderPass2](#)
- [vkCmdNextSubpass2](#)
- [vkCreateRenderPass2](#)
- [vkGetBufferDeviceAddress](#)
- [vkGetBufferOpaqueCaptureAddress](#)
- [vkGetDeviceMemoryOpaqueCaptureAddress](#)
- [vkGetSemaphoreCounterValue](#)
- [vkResetQueryPool](#)
- [vkSignalSemaphore](#)
- [vkWaitSemaphores](#)

New Structures

- [VkAttachmentDescription2](#)
- [VkAttachmentReference2](#)
- [VkBufferDeviceAddressInfo](#)
- [VkConformanceVersion](#)
- [VkDeviceMemoryOpaqueCaptureAddressInfo](#)
- [VkFramebufferAttachmentImageInfo](#)
- [VkRenderPassCreateInfo2](#)
- [VkSemaphoreSignalInfo](#)
- [VkSemaphoreWaitInfo](#)
- [VkSubpassBeginInfo](#)
- [VkSubpassDependency2](#)
- [VkSubpassDescription2](#)
- [VkSubpassEndInfo](#)
- Extending [VkAttachmentDescription2](#):
 - [VkAttachmentDescriptionStencilLayout](#)
- Extending [VkAttachmentReference2](#):
 - [VkAttachmentReferenceStencilLayout](#)
- Extending [VkBufferCreateInfo](#):

- [VkBufferOpaqueCaptureAddressCreateInfo](#)
- Extending [VkDescriptorSetAllocateInfo](#):
 - [VkDescriptorSetVariableDescriptorCountAllocateInfo](#)
- Extending [VkDescriptorSetLayoutCreateInfo](#):
 - [VkDescriptorSetLayoutBindingFlagsCreateInfo](#)
- Extending [VkDescriptorSetLayoutSupport](#):
 - [VkDescriptorSetVariableDescriptorCountLayoutSupport](#)
- Extending [VkFramebufferCreateInfo](#):
 - [VkFramebufferAttachmentsCreateInfo](#)
- Extending [VkImageCreateInfo](#), [VkPhysicalDeviceImageFormatInfo2](#):
 - [VkImageStencilUsageCreateInfo](#)
- Extending [VkImageCreateInfo](#), [VkPhysicalDeviceImageFormatInfo2](#), [VkSwapchainCreateInfoKHR](#):
 - [VkImageFormatListCreateInfo](#)
- Extending [VkMemoryAllocateInfo](#):
 - [VkMemoryOpaqueCaptureAddressAllocateInfo](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDevice8BitStorageFeatures](#)
 - [VkPhysicalDeviceBufferDeviceAddressFeatures](#)
 - [VkPhysicalDeviceDescriptorIndexingFeatures](#)
 - [VkPhysicalDeviceHostQueryResetFeatures](#)
 - [VkPhysicalDeviceImagelessFramebufferFeatures](#)
 - [VkPhysicalDeviceScalarBlockLayoutFeatures](#)
 - [VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures](#)
 - [VkPhysicalDeviceShaderAtomicInt64Features](#)
 - [VkPhysicalDeviceShaderFloat16Int8Features](#)
 - [VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures](#)
 - [VkPhysicalDeviceTimelineSemaphoreFeatures](#)
 - [VkPhysicalDeviceUniformBufferStandardLayoutFeatures](#)
 - [VkPhysicalDeviceVulkan11Features](#)
 - [VkPhysicalDeviceVulkan12Features](#)
 - [VkPhysicalDeviceVulkanMemoryModelFeatures](#)
- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDeviceDepthStencilResolveProperties](#)
 - [VkPhysicalDeviceDescriptorIndexingProperties](#)

- [VkPhysicalDeviceDriverProperties](#)
- [VkPhysicalDeviceFloatControlsProperties](#)
- [VkPhysicalDeviceSamplerFilterMinmaxProperties](#)
- [VkPhysicalDeviceTimelineSemaphoreProperties](#)
- [VkPhysicalDeviceVulkan11Properties](#)
- [VkPhysicalDeviceVulkan12Properties](#)
- Extending [VkRenderPassBeginInfo](#):
 - [VkRenderPassAttachmentBeginInfo](#)
- Extending [VkSamplerCreateInfo](#):
 - [VkSamplerReductionModeCreateInfo](#)
- Extending [VkSemaphoreCreateInfo](#), [VkPhysicalDeviceExternalSemaphoreInfo](#):
 - [VkSemaphoreTypeCreateInfo](#)
- Extending [VkSubmitInfo](#), [VkBindSparseInfo](#):
 - [VkTimelineSemaphoreSubmitInfo](#)
- Extending [VkSubpassDescription2](#):
 - [VkSubpassDescriptionDepthStencilResolve](#)

New Enums

- [VkDescriptorBindingFlagBits](#)
- [VkDriverId](#)
- [VkResolveModeFlagBits](#)
- [VkSamplerReductionMode](#)
- [VkSemaphoreType](#)
- [VkSemaphoreWaitFlagBits](#)
- [VkShaderFloatControlsIndependence](#)

New Bitmasks

- [VkDescriptorBindingFlags](#)
- [VkResolveModeFlags](#)
- [VkSemaphoreWaitFlags](#)

New Enum Constants

- [VK_MAX_DRIVER_INFO_SIZE](#)
- [VK_MAX_DRIVER_NAME_SIZE](#)
- Extending [VkBufferCreateFlagBits](#):

- `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`
- Extending `VkBufferUsageFlagBits`:
 - `VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT`
- Extending `VkDescriptorPoolCreateFlagBits`:
 - `VK_DESCRIPTOR_POOL_CREATE_UPDATE_AFTER_BIND_BIT`
- Extending `VkDescriptorSetLayoutCreateFlagBits`:
 - `VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT`
- Extending `VkFormatFeatureFlagBits`:
 - `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_MINMAX_BIT`
- Extending `VkFramebufferCreateFlagBits`:
 - `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`
- Extending `VkImageLayout`:
 - `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`
 - `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`
 - `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL`
 - `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- Extending `VkMemoryAllocateFlagBits`:
 - `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_BIT`
 - `VK_MEMORY_ALLOCATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`
- Extending `VkResult`:
 - `VK_ERROR_FRAGMENTATION`
 - `VK_ERROR_INVALID_OPAQUE_CAPTURE_ADDRESS`
- Extending `VkSamplerAddressMode`:
 - `VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_2`
 - `VK_STRUCTURE_TYPE_ATTACHMENT_DESCRIPTION_STENCIL_LAYOUT`
 - `VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_2`
 - `VK_STRUCTURE_TYPE_ATTACHMENT_REFERENCE_STENCIL_LAYOUT`
 - `VK_STRUCTURE_TYPE_BUFFER_DEVICE_ADDRESS_INFO`
 - `VK_STRUCTURE_TYPE_BUFFER_OPAQUE_CAPTURE_ADDRESS_CREATE_INFO`
 - `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_BINDING_FLAGS_CREATE_INFO`
 - `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_ALLOCATE_INFO`
 - `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_VARIABLE_DESCRIPTOR_COUNT_LAYOUT_SUPPORT`
 - `VK_STRUCTURE_TYPE_DEVICE_MEMORY_OPAQUE_CAPTURE_ADDRESS_INFO`

- VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENTS_CREATE_INFO
- VK_STRUCTURE_TYPE_FRAMEBUFFER_ATTACHMENT_IMAGE_INFO
- VK_STRUCTURE_TYPE_IMAGE_FORMAT_LIST_CREATE_INFO
- VK_STRUCTURE_TYPE_IMAGE_STENCIL_USAGE_CREATE_INFO
- VK_STRUCTURE_TYPE_MEMORY_OPAQUE_CAPTURE_ADDRESS_ALLOCATE_INFO
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_8BIT_STORAGE_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BUFFER_DEVICE_ADDRESS_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_STENCIL_RESOLVE_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DESCRIPTOR_INDEXING_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DRIVER_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FLOAT_CONTROLS_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_HOST_QUERY_RESET_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGELESS_FRAMEBUFFER_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_FILTER_MINMAX_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SCALAR_BLOCK_LAYOUT_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SEPARATE_DEPTH_STENCIL_LAYOUTS_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_INT64_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_FLOAT16_INT8_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_SUBGROUP_EXTENDED_TYPES_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TIMELINE_SEMAPHORE_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_UNIFORM_BUFFER_STANDARD_LAYOUT_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_1_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_1_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_2_FEATURES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_2_PROPERTIES
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_MEMORY_MODEL_FEATURES
- VK_STRUCTURE_TYPE_RENDER_PASS_ATTACHMENT_BEGIN_INFO
- VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO_2
- VK_STRUCTURE_TYPE_SAMPLER_REDUCTION_MODE_CREATE_INFO
- VK_STRUCTURE_TYPE_SEMAPHORE_SIGNAL_INFO
- VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO
- VK_STRUCTURE_TYPE_SEMAPHORE_WAIT_INFO
- VK_STRUCTURE_TYPE_SUBPASS_BEGIN_INFO

- `VK_STRUCTURE_TYPE_SUBPASS_DEPENDENCY_2`
- `VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_2`
- `VK_STRUCTURE_TYPE_SUBPASS_DESCRIPTION_DEPTH_STENCIL_RESOLVE`
- `VK_STRUCTURE_TYPE_SUBPASS_END_INFO`
- `VK_STRUCTURE_TYPE_TIMELINE_SEMAPHORE_SUBMIT_INFO`

Vulkan Version 1.1

Vulkan Version 1.1 [promoted](#) a number of key extensions into the core API:

- `VK_KHR_16bit_storage`
- `VK_KHR_bind_memory2`
- `VK_KHR_dedicated_allocation`
- `VK_KHR_descriptor_update_template`
- `VK_KHR_device_group`
- `VK_KHR_device_group_creation`
- `VK_KHR_external_fence`
- `VK_KHR_external_fence_capabilities`
- `VK_KHR_external_memory`
- `VK_KHR_external_memory_capabilities`
- `VK_KHR_external_semaphore`
- `VK_KHR_external_semaphore_capabilities`
- `VK_KHR_get_memory_requirements2`
- `VK_KHR_get_physical_device_properties2`
- `VK_KHR_maintenance1`
- `VK_KHR_maintenance2`
- `VK_KHR_maintenance3`
- `VK_KHR_multiview`
- `VK_KHR_relaxed_block_layout`
- `VK_KHR_sampler_ycbcr_conversion`
- `VK_KHR_shader_draw_parameters`
- `VK_KHR_storage_buffer_storage_class`
- `VK_KHR_variable_pointers`

All differences in behavior between these extensions and the corresponding Vulkan 1.1 functionality are summarized below.

Differences Relative to `VK_KHR_16bit_storage`

If the `VK_KHR_16bit_storage` extension is not supported, support for the `storageBuffer16BitAccess` feature is optional. Support for this feature is defined by `VkPhysicalDevice16BitStorageFeatures::storageBuffer16BitAccess` or `VkPhysicalDeviceVulkan11Features::storageBuffer16BitAccess` when queried via `vkGetPhysicalDeviceFeatures2`.

Differences Relative to `VK_KHR_sampler_ycbcr_conversion`

If the `VK_KHR_sampler_ycbcr_conversion` extension is not supported, support for the `samplerYcbcrConversion` feature is optional. Support for this feature is defined by `VkPhysicalDeviceSamplerYcbcrConversionFeatures::samplerYcbcrConversion` or `VkPhysicalDeviceVulkan11Features::samplerYcbcrConversion` when queried via `vkGetPhysicalDeviceFeatures2`.

Differences Relative to `VK_KHR_shader_draw_parameters`

If the `VK_KHR_shader_draw_parameters` extension is not supported, support for the `SPV_KHR_shader_draw_parameters` SPIR-V extension is optional. Support for this feature is defined by `VkPhysicalDeviceShaderDrawParametersFeatures::shaderDrawParameters` or `VkPhysicalDeviceVulkan11Features::shaderDrawParameters` when queried via `vkGetPhysicalDeviceFeatures2`.

Differences Relative to `VK_KHR_variable_pointers`

If the `VK_KHR_variable_pointers` extension is not supported, support for the `variablePointersStorageBuffer` feature is optional. Support for this feature is defined by `VkPhysicalDeviceVariablePointersFeatures::variablePointersStorageBuffer` or `VkPhysicalDeviceVulkan11Features::variablePointersStorageBuffer` when queried via `vkGetPhysicalDeviceFeatures2`.

Additional Vulkan 1.1 Feature Support

In addition to the promoted extensions described above, Vulkan 1.1 added support for:

- SPIR-V version 1.1
- SPIR-V version 1.2
- SPIR-V version 1.3
- The `group operations` and `subgroup scope`.
- The `protected memory` feature.
- A new command to enumerate the instance version: `vkEnumerateInstanceVersion`.
- The `VkPhysicalDeviceShaderDrawParametersFeatures` feature query structure (where the `VK_KHR_shader_draw_parameters` extension did not have one).

New Macros

- `VK_API_VERSION_1_1`

New Object Types

- `VkDescriptorUpdateTemplate`

- [VkSamplerYcbcrConversion](#)

New Commands

- [vkBindBufferMemory2](#)
- [vkBindImageMemory2](#)
- [vkCmdDispatchBase](#)
- [vkCmdSetDeviceMask](#)
- [vkCreateDescriptorUpdateTemplate](#)
- [vkCreateSamplerYcbcrConversion](#)
- [vkDestroyDescriptorUpdateTemplate](#)
- [vkDestroySamplerYcbcrConversion](#)
- [vkEnumerateInstanceVersion](#)
- [vkEnumeratePhysicalDeviceGroups](#)
- [vkGetBufferMemoryRequirements2](#)
- [vkGetDescriptorSetLayoutSupport](#)
- [vkGetDeviceGroupPeerMemoryFeatures](#)
- [vkGetDeviceQueue2](#)
- [vkGetImageMemoryRequirements2](#)
- [vkGetImageSparseMemoryRequirements2](#)
- [vkGetPhysicalDeviceExternalBufferProperties](#)
- [vkGetPhysicalDeviceExternalFenceProperties](#)
- [vkGetPhysicalDeviceExternalSemaphoreProperties](#)
- [vkGetPhysicalDeviceFeatures2](#)
- [vkGetPhysicalDeviceFormatProperties2](#)
- [vkGetPhysicalDeviceImageFormatProperties2](#)
- [vkGetPhysicalDeviceMemoryProperties2](#)
- [vkGetPhysicalDeviceProperties2](#)
- [vkGetPhysicalDeviceQueueFamilyProperties2](#)
- [vkGetPhysicalDeviceSparseImageFormatProperties2](#)
- [vkTrimCommandPool](#)
- [vkUpdateDescriptorSetWithTemplate](#)

New Structures

- [VkBindBufferMemoryInfo](#)
- [VkBindImageMemoryInfo](#)

- [VkBufferMemoryRequirementsInfo2](#)
- [VkDescriptorSetLayoutSupport](#)
- [VkDescriptorUpdateTemplateCreateInfo](#)
- [VkDescriptorUpdateTemplateEntry](#)
- [VkDeviceQueueInfo2](#)
- [VkExternalBufferProperties](#)
- [VkExternalFenceProperties](#)
- [VkExternalMemoryProperties](#)
- [VkExternalSemaphoreProperties](#)
- [VkFormatProperties2](#)
- [VkImageFormatProperties2](#)
- [VkImageMemoryRequirementsInfo2](#)
- [VkImageSparseMemoryRequirementsInfo2](#)
- [VkInputAttachmentAspectReference](#)
- [VkMemoryRequirements2](#)
- [VkPhysicalDeviceExternalBufferInfo](#)
- [VkPhysicalDeviceExternalFenceInfo](#)
- [VkPhysicalDeviceExternalSemaphoreInfo](#)
- [VkPhysicalDeviceGroupProperties](#)
- [VkPhysicalDeviceImageFormatInfo2](#)
- [VkPhysicalDeviceMemoryProperties2](#)
- [VkPhysicalDeviceProperties2](#)
- [VkPhysicalDeviceSparseImageFormatInfo2](#)
- [VkQueueFamilyProperties2](#)
- [VkSamplerYcbcrConversionCreateInfo](#)
- [VkSparseImageFormatProperties2](#)
- [VkSparseImageMemoryRequirements2](#)
- Extending [VkBindBufferMemoryInfo](#):
 - [VkBindBufferMemoryDeviceGroupInfo](#)
- Extending [VkBindImageMemoryInfo](#):
 - [VkBindImageMemoryDeviceGroupInfo](#)
 - [VkBindImagePlaneMemoryInfo](#)
- Extending [VkBindSparseInfo](#):
 - [VkDeviceGroupBindSparseInfo](#)
- Extending [VkBufferCreateInfo](#):

- [VkExternalMemoryBufferCreateInfo](#)
- Extending [VkCommandBufferBeginInfo](#):
 - [VkDeviceGroupCommandBufferBeginInfo](#)
- Extending [VkDeviceCreateInfo](#):
 - [VkDeviceGroupDeviceCreateInfo](#)
 - [VkPhysicalDeviceFeatures2](#)
- Extending [VkFenceCreateInfo](#):
 - [VkExportFenceCreateInfo](#)
- Extending [VkImageCreateInfo](#):
 - [VkExternalMemoryImageCreateInfo](#)
- Extending [VkImageFormatProperties2](#):
 - [VkExternalImageFormatProperties](#)
 - [VkSamplerYcbcrConversionImageFormatProperties](#)
- Extending [VkImageMemoryRequirementsInfo2](#):
 - [VkImagePlaneMemoryRequirementsInfo](#)
- Extending [VkImageViewCreateInfo](#):
 - [VkImageViewUsageCreateInfo](#)
- Extending [VkMemoryAllocateInfo](#):
 - [VkExportMemoryAllocateInfo](#)
 - [VkMemoryAllocateFlagsInfo](#)
 - [VkMemoryDedicatedAllocateInfo](#)
- Extending [VkMemoryRequirements2](#):
 - [VkMemoryDedicatedRequirements](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDevice16BitStorageFeatures](#)
 - [VkPhysicalDeviceMultiviewFeatures](#)
 - [VkPhysicalDeviceProtectedMemoryFeatures](#)
 - [VkPhysicalDeviceSamplerYcbcrConversionFeatures](#)
 - [VkPhysicalDeviceShaderDrawParameterFeatures](#)
 - [VkPhysicalDeviceShaderDrawParametersFeatures](#)
 - [VkPhysicalDeviceVariablePointerFeatures](#)
 - [VkPhysicalDeviceVariablePointersFeatures](#)
- Extending [VkPhysicalDeviceImageFormatInfo2](#):
 - [VkPhysicalDeviceExternalImageFormatInfo](#)
- Extending [VkPhysicalDeviceProperties2](#):

- `VkPhysicalDeviceIDProperties`
- `VkPhysicalDeviceMaintenance3Properties`
- `VkPhysicalDeviceMultiviewProperties`
- `VkPhysicalDevicePointClippingProperties`
- `VkPhysicalDeviceProtectedMemoryProperties`
- `VkPhysicalDeviceSubgroupProperties`
- Extending `VkPipelineTessellationStateCreateInfo`:
 - `VkPipelineTessellationDomainOriginStateCreateInfo`
- Extending `VkRenderPassBeginInfo`, `VkRenderingInfo`:
 - `VkDeviceGroupRenderPassBeginInfo`
- Extending `VkRenderPassCreateInfo`:
 - `VkRenderPassInputAttachmentAspectCreateInfo`
 - `VkRenderPassMultiviewCreateInfo`
- Extending `VkSamplerCreateInfo`, `VkImageViewCreateInfo`:
 - `VkSamplerYcbcrConversionInfo`
- Extending `VkSemaphoreCreateInfo`:
 - `VkExportSemaphoreCreateInfo`
- Extending `VkSubmitInfo`:
 - `VkDeviceGroupSubmitInfo`
 - `VkProtectedSubmitInfo`

New Enums

- `VkChromaLocation`
- `VkDescriptorUpdateTemplateType`
- `VkDeviceQueueCreateFlagBits`
- `VkExternalFenceFeatureFlagBits`
- `VkExternalFenceHandleTypeFlagBits`
- `VkExternalMemoryFeatureFlagBits`
- `VkExternalMemoryHandleTypeFlagBits`
- `VkExternalSemaphoreFeatureFlagBits`
- `VkExternalSemaphoreHandleTypeFlagBits`
- `VkFenceImportFlagBits`
- `VkMemoryAllocateFlagBits`
- `VkPeerMemoryFeatureFlagBits`
- `VkPointClippingBehavior`

- [VkSamplerYcbcrModelConversion](#)
- [VkSamplerYcbcrRange](#)
- [VkSemaphoreImportFlagBits](#)
- [VkSubgroupFeatureFlagBits](#)
- [VkTessellationDomainOrigin](#)

New Bitmasks

- [VkCommandPoolTrimFlags](#)
- [VkDescriptorUpdateTemplateCreateFlags](#)
- [VkExternalFenceFeatureFlags](#)
- [VkExternalFenceHandleTypeFlags](#)
- [VkExternalMemoryFeatureFlags](#)
- [VkExternalMemoryHandleTypeFlags](#)
- [VkExternalSemaphoreFeatureFlags](#)
- [VkExternalSemaphoreHandleTypeFlags](#)
- [VkFenceImportFlags](#)
- [VkMemoryAllocateFlags](#)
- [VkPeerMemoryFeatureFlags](#)
- [VkSemaphoreImportFlags](#)
- [VkSubgroupFeatureFlags](#)

New Enum Constants

- [VK_LUID_SIZE](#)
- [VK_MAX_DEVICE_GROUP_SIZE](#)
- [VK_QUEUE_FAMILY_EXTERNAL](#)
- Extending [VkBufferCreateFlagBits](#):
 - [VK_BUFFER_CREATE_PROTECTED_BIT](#)
- Extending [VkCommandPoolCreateFlagBits](#):
 - [VK_COMMAND_POOL_CREATE_PROTECTED_BIT](#)
- Extending [VkDependencyFlagBits](#):
 - [VK_DEPENDENCY_DEVICE_GROUP_BIT](#)
 - [VK_DEPENDENCY_VIEW_LOCAL_BIT](#)
- Extending [VkDeviceQueueCreateFlagBits](#):
 - [VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT](#)
- Extending [VkFormat](#):

- VK_FORMAT_B10X6G10X6R10X6G10X6_422_UNORM_4PACK16
- VK_FORMAT_B12X4G12X4R12X4G12X4_422_UNORM_4PACK16
- VK_FORMAT_B16G16R16G16_422_UNORM
- VK_FORMAT_B8G8R8G8_422_UNORM
- VK_FORMAT_G10X6B10X6G10X6R10X6_422_UNORM_4PACK16
- VK_FORMAT_G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16
- VK_FORMAT_G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16
- VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16
- VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16
- VK_FORMAT_G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16
- VK_FORMAT_G12X4B12X4G12X4R12X4_422_UNORM_4PACK16
- VK_FORMAT_G12X4_B12X4R12X4_2PLANE_420_UNORM_3PACK16
- VK_FORMAT_G12X4_B12X4R12X4_2PLANE_422_UNORM_3PACK16
- VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16
- VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16
- VK_FORMAT_G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16
- VK_FORMAT_G16B16G16R16_422_UNORM
- VK_FORMAT_G16_B16R16_2PLANE_420_UNORM
- VK_FORMAT_G16_B16R16_2PLANE_422_UNORM
- VK_FORMAT_G16_B16_R16_3PLANE_420_UNORM
- VK_FORMAT_G16_B16_R16_3PLANE_422_UNORM
- VK_FORMAT_G16_B16_R16_3PLANE_444_UNORM
- VK_FORMAT_G8B8G8R8_422_UNORM
- VK_FORMAT_G8_B8R8_2PLANE_420_UNORM
- VK_FORMAT_G8_B8R8_2PLANE_422_UNORM
- VK_FORMAT_G8_B8_R8_3PLANE_420_UNORM
- VK_FORMAT_G8_B8_R8_3PLANE_422_UNORM
- VK_FORMAT_G8_B8_R8_3PLANE_444_UNORM
- VK_FORMAT_R10X6G10X6B10X6A10X6_UNORM_4PACK16
- VK_FORMAT_R10X6G10X6_UNORM_2PACK16
- VK_FORMAT_R10X6_UNORM_PACK16
- VK_FORMAT_R12X4G12X4B12X4A12X4_UNORM_4PACK16
- VK_FORMAT_R12X4G12X4_UNORM_2PACK16
- VK_FORMAT_R12X4_UNORM_PACK16

- Extending [VkFormatFeatureFlagBits](#):

- VK_FORMAT_FEATURE_COSITED_CHROMA_SAMPLES_BIT
- VK_FORMAT_FEATURE_DISJOINT_BIT
- VK_FORMAT_FEATURE_MIDPOINT_CHROMA_SAMPLES_BIT
- VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_BIT
- VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE_BIT
- VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_LINEAR_FILTER_BIT
- VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_CONVERSION_SEPARATE_RECONSTRUCTION_FILTER_BIT
- VK_FORMAT_FEATURE_TRANSFER_DST_BIT
- VK_FORMAT_FEATURE_TRANSFER_SRC_BIT
- Extending [VkImageAspectFlagBits](#):
 - VK_IMAGE_ASPECT_PLANE_0_BIT
 - VK_IMAGE_ASPECT_PLANE_1_BIT
 - VK_IMAGE_ASPECT_PLANE_2_BIT
- Extending [VkImageCreateFlagBits](#):
 - VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT
 - VK_IMAGE_CREATE_ALIAS_BIT
 - VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT
 - VK_IMAGE_CREATE_DISJOINT_BIT
 - VK_IMAGE_CREATE_EXTENDED_USAGE_BIT
 - VK_IMAGE_CREATE_PROTECTED_BIT
 - VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT
- Extending [VkImageLayout](#):
 - VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL
 - VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- Extending [VkMemoryHeapFlagBits](#):
 - VK_MEMORY_HEAP_MULTI_INSTANCE_BIT
- Extending [VkMemoryPropertyFlagBits](#):
 - VK_MEMORY_PROPERTY_PROTECTED_BIT
- Extending [VkObjectType](#):
 - VK_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE
 - VK_OBJECT_TYPE_SAMPLER_YCBCR_CONVERSION
- Extending [VkPipelineCreateFlagBits](#):
 - VK_PIPELINE_CREATE_DISPATCH_BASE
 - VK_PIPELINE_CREATE_DISPATCH_BASE_BIT

- VK_PIPELINE_CREATE_VIEW_INDEX_FROM_DEVICE_INDEX_BIT
- Extending [VkQueueFlagBits](#):
 - VK_QUEUE_PROTECTED_BIT
- Extending [VkResult](#):
 - VK_ERROR_INVALID_EXTERNAL_HANDLE
 - VK_ERROR_OUT_OF_POOL_MEMORY
- Extending [VkStructureType](#):
 - VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_DEVICE_GROUP_INFO
 - VK_STRUCTURE_TYPE_BIND_BUFFER_MEMORY_INFO
 - VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_DEVICE_GROUP_INFO
 - VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_INFO
 - VK_STRUCTURE_TYPE_BIND_IMAGE_PLANE_MEMORY_INFO
 - VK_STRUCTURE_TYPE_BUFFER_MEMORY_REQUIREMENTS_INFO_2
 - VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_SUPPORT
 - VK_STRUCTURE_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_CREATE_INFO
 - VK_STRUCTURE_TYPE_DEVICE_GROUP_BIND_SPARSE_INFO
 - VK_STRUCTURE_TYPE_DEVICE_GROUP_COMMAND_BUFFER_BEGIN_INFO
 - VK_STRUCTURE_TYPE_DEVICE_GROUP_DEVICE_CREATE_INFO
 - VK_STRUCTURE_TYPE_DEVICE_GROUP_RENDER_PASS_BEGIN_INFO
 - VK_STRUCTURE_TYPE_DEVICE_GROUP_SUBMIT_INFO
 - VK_STRUCTURE_TYPE_DEVICE_QUEUE_INFO_2
 - VK_STRUCTURE_TYPE_EXPORT_FENCE_CREATE_INFO
 - VK_STRUCTURE_TYPE_EXPORT_MEMORY_ALLOCATE_INFO
 - VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_CREATE_INFO
 - VK_STRUCTURE_TYPE_EXTERNAL_BUFFER_PROPERTIES
 - VK_STRUCTURE_TYPE_EXTERNAL_FENCE_PROPERTIES
 - VK_STRUCTURE_TYPE_EXTERNAL_IMAGE_FORMAT_PROPERTIES
 - VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_BUFFER_CREATE_INFO
 - VK_STRUCTURE_TYPE_EXTERNAL_MEMORY_IMAGE_CREATE_INFO
 - VK_STRUCTURE_TYPE_EXTERNAL_SEMAPHORE_PROPERTIES
 - VK_STRUCTURE_TYPE_FORMAT_PROPERTIES_2
 - VK_STRUCTURE_TYPE_IMAGE_FORMAT_PROPERTIES_2
 - VK_STRUCTURE_TYPE_IMAGE_MEMORY_REQUIREMENTS_INFO_2
 - VK_STRUCTURE_TYPE_IMAGE_PLANE_MEMORY_REQUIREMENTS_INFO
 - VK_STRUCTURE_TYPE_IMAGE_SPARSE_MEMORY_REQUIREMENTS_INFO_2

- `VK_STRUCTURE_TYPE_IMAGE_VIEW_USAGE_CREATE_INFO`
- `VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_FLAGS_INFO`
- `VK_STRUCTURE_TYPE_MEMORY_DEDICATED_ALLOCATE_INFO`
- `VK_STRUCTURE_TYPE_MEMORY_DEDICATED_REQUIREMENTS`
- `VK_STRUCTURE_TYPE_MEMORY_REQUIREMENTS_2`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_16BIT_STORAGE_FEATURES`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_BUFFER_INFO`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_FENCE_INFO`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_IMAGE_FORMAT_INFO`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SEMAPHORE_INFO`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FEATURES_2`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GROUP_PROPERTIES`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ID_PROPERTIES`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_FORMAT_INFO_2`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MAINTENANCE_3_PROPERTIES`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_PROPERTIES_2`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_FEATURES`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MULTIVIEW_PROPERTIES`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_POINT_CLIPPING_PROPERTIES`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROPERTIES_2`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROTECTED_MEMORY_FEATURES`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PROTECTED_MEMORY_PROPERTIES`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLER_YCBCR_CONVERSION_FEATURES`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DRAW_PARAMETERS_FEATURES`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SPARSE_IMAGE_FORMAT_INFO_2`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_PROPERTIES`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTERS_FEATURES`
- `VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_DOMAIN_ORIGIN_STATE_CREATE_INFO`
- `VK_STRUCTURE_TYPE_PROTECTED_SUBMIT_INFO`
- `VK_STRUCTURE_TYPE_QUEUE_FAMILY_PROPERTIES_2`
- `VK_STRUCTURE_TYPE_RENDER_PASS_INPUT_ATTACHMENT_ASPECT_CREATE_INFO`
- `VK_STRUCTURE_TYPE_RENDER_PASS_MULTIVIEW_CREATE_INFO`
- `VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_CREATE_INFO`
- `VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_IMAGE_FORMAT_PROPERTIES`
- `VK_STRUCTURE_TYPE_SAMPLER_YCBCR_CONVERSION_INFO`

- [VK_STRUCTURE_TYPE_SPARSE_IMAGE_FORMAT_PROPERTIES_2](#)
- [VK_STRUCTURE_TYPE_SPARSE_IMAGE_MEMORY_REQUIREMENTS_2](#)

Vulkan Version 1.0

Vulkan Version 1.0 was the initial release of the Vulkan API.

New Macros

- [VK_API_VERSION](#)
- [VK_API_VERSION_1_0](#)
- [VK_API_VERSION_MAJOR](#)
- [VK_API_VERSION_MINOR](#)
- [VK_API_VERSION_PATCH](#)
- [VK_API_VERSION_VARIANT](#)
- [VK_DEFINE_HANDLE](#)
- [VK_DEFINE_NON_DISPATCHABLE_HANDLE](#)
- [VK_HEADER_VERSION](#)
- [VK_HEADER_VERSION_COMPLETE](#)
- [VK_MAKE_API_VERSION](#)
- [VK_MAKE_VERSION](#)
- [VK_NULL_HANDLE](#)
- [VK_USE_64_BIT_PTR_DEFINES](#)
- [VK_VERSION_MAJOR](#)
- [VK_VERSION_MINOR](#)
- [VK_VERSION_PATCH](#)

New Base Types

- [VkBool32](#)
- [VkDeviceAddress](#)
- [VkDeviceSize](#)
- [VkFlags](#)
- [VkSampleMask](#)

New Object Types

- [VkBuffer](#)
- [VkBufferView](#)

- [VkCommandBuffer](#)
- [VkCommandPool](#)
- [VkDescriptorPool](#)
- [VkDescriptorSet](#)
- [VkDescriptorSetLayout](#)
- [VkDevice](#)
- [VkDeviceMemory](#)
- [VkEvent](#)
- [VkFence](#)
- [VkFramebuffer](#)
- [VkImage](#)
- [VkImageView](#)
- [VkInstance](#)
- [VkPhysicalDevice](#)
- [VkPipeline](#)
- [VkPipelineCache](#)
- [VkPipelineLayout](#)
- [VkQueryPool](#)
- [VkQueue](#)
- [VkRenderPass](#)
- [VkSampler](#)
- [VkSemaphore](#)
- [VkShaderModule](#)

New Commands

- [vkAllocateCommandBuffers](#)
- [vkAllocateDescriptorSets](#)
- [vkAllocateMemory](#)
- [vkBeginCommandBuffer](#)
- [vkBindBufferMemory](#)
- [vkBindImageMemory](#)
- [vkCmdBeginQuery](#)
- [vkCmdBeginRenderPass](#)
- [vkCmdBindDescriptorSets](#)
- [vkCmdBindIndexBuffer](#)

- [vkCmdBindPipeline](#)
- [vkCmdBindVertexBuffers](#)
- [vkCmdBlitImage](#)
- [vkCmdClearAttachments](#)
- [vkCmdClearColorImage](#)
- [vkCmdClearDepthStencilImage](#)
- [vkCmdCopyBuffer](#)
- [vkCmdCopyBufferToImage](#)
- [vkCmdCopyImage](#)
- [vkCmdCopyImageToBuffer](#)
- [vkCmdCopyQueryPoolResults](#)
- [vkCmdDispatch](#)
- [vkCmdDispatchIndirect](#)
- [vkCmdDraw](#)
- [vkCmdDrawIndexed](#)
- [vkCmdDrawIndexedIndirect](#)
- [vkCmdDrawIndirect](#)
- [vkCmdEndQuery](#)
- [vkCmdEndRenderPass](#)
- [vkCmdExecuteCommands](#)
- [vkCmdFillBuffer](#)
- [vkCmdNextSubpass](#)
- [vkCmdPipelineBarrier](#)
- [vkCmdPushConstants](#)
- [vkCmdResetEvent](#)
- [vkCmdResetQueryPool](#)
- [vkCmdResolveImage](#)
- [vkCmdSetBlendConstants](#)
- [vkCmdSetDepthBias](#)
- [vkCmdSetDepthBounds](#)
- [vkCmdSetEvent](#)
- [vkCmdSetLineWidth](#)
- [vkCmdSetScissor](#)
- [vkCmdSetStencilCompareMask](#)
- [vkCmdSetStencilReference](#)

- `vkCmdSetStencilWriteMask`
- `vkCmdSetViewport`
- `vkCmdUpdateBuffer`
- `vkCmdWaitEvents`
- `vkCmdWriteTimestamp`
- `vkCreateBuffer`
- `vkCreateBufferView`
- `vkCreateCommandPool`
- `vkCreateComputePipelines`
- `vkCreateDescriptorPool`
- `vkCreateDescriptorSetLayout`
- `vkCreateDevice`
- `vkCreateEvent`
- `vkCreateFence`
- `vkCreateFramebuffer`
- `vkCreateGraphicsPipelines`
- `vkCreateImage`
- `vkCreateImageView`
- `vkCreateInstance`
- `vkCreatePipelineCache`
- `vkCreatePipelineLayout`
- `vkCreateQueryPool`
- `vkCreateRenderPass`
- `vkCreateSampler`
- `vkCreateSemaphore`
- `vkCreateShaderModule`
- `vkDestroyBuffer`
- `vkDestroyBufferView`
- `vkDestroyCommandPool`
- `vkDestroyDescriptorPool`
- `vkDestroyDescriptorSetLayout`
- `vkDestroyDevice`
- `vkDestroyEvent`
- `vkDestroyFence`
- `vkDestroyFramebuffer`

- [vkDestroyImage](#)
- [vkDestroyImageView](#)
- [vkDestroyInstance](#)
- [vkDestroyPipeline](#)
- [vkDestroyPipelineCache](#)
- [vkDestroyPipelineLayout](#)
- [vkDestroyQueryPool](#)
- [vkDestroyRenderPass](#)
- [vkDestroySampler](#)
- [vkDestroySemaphore](#)
- [vkDestroyShaderModule](#)
- [vkDeviceWaitIdle](#)
- [vkEndCommandBuffer](#)
- [vkEnumerateDeviceExtensionProperties](#)
- [vkEnumerateDeviceLayerProperties](#)
- [vkEnumerateInstanceExtensionProperties](#)
- [vkEnumerateInstanceLayerProperties](#)
- [vkEnumeratePhysicalDevices](#)
- [vkFlushMappedMemoryRanges](#)
- [vkFreeCommandBuffers](#)
- [vkFreeDescriptorSets](#)
- [vkFreeMemory](#)
- [vkGetBufferMemoryRequirements](#)
- [vkGetDeviceMemoryCommitment](#)
- [vkGetDeviceProcAddr](#)
- [vkGetDeviceQueue](#)
- [vkGetEventStatus](#)
- [vkGetFenceStatus](#)
- [vkGetImageMemoryRequirements](#)
- [vkGetImageSparseMemoryRequirements](#)
- [vkGetImageSubresourceLayout](#)
- [vkGetInstanceProcAddr](#)
- [vkGetPhysicalDeviceFeatures](#)
- [vkGetPhysicalDeviceFormatProperties](#)
- [vkGetPhysicalDeviceImageFormatProperties](#)

- [vkGetPhysicalDeviceMemoryProperties](#)
- [vkGetPhysicalDeviceProperties](#)
- [vkGetPhysicalDeviceQueueFamilyProperties](#)
- [vkGetPhysicalDeviceSparseImageFormatProperties](#)
- [vkGetPipelineCacheData](#)
- [vkGetQueryPoolResults](#)
- [vkGetRenderAreaGranularity](#)
- [vkInvalidateMappedMemoryRanges](#)
- [vkMapMemory](#)
- [vkMergePipelineCaches](#)
- [vkQueueBindSparse](#)
- [vkQueueSubmit](#)
- [vkQueueWaitIdle](#)
- [vkResetCommandBuffer](#)
- [vkResetCommandPool](#)
- [vkResetDescriptorPool](#)
- [vkResetEvent](#)
- [vkResetFences](#)
- [vkSetEvent](#)
- [vkUnmapMemory](#)
- [vkUpdateDescriptorSets](#)
- [vkWaitForFences](#)

New Structures

- [VkAllocationCallbacks](#)
- [VkApplicationInfo](#)
- [VkAttachmentDescription](#)
- [VkAttachmentReference](#)
- [VkBaseInStructure](#)
- [VkBaseOutStructure](#)
- [VkBindSparseInfo](#)
- [VkBufferCopy](#)
- [VkBufferCreateInfo](#)
- [VkBufferImageCopy](#)
- [VkBufferMemoryBarrier](#)

- [VkBufferViewCreateInfo](#)
- [VkClearAttachment](#)
- [VkClearDepthStencilValue](#)
- [VkClearRect](#)
- [VkCommandBufferAllocateInfo](#)
- [VkCommandBufferBeginInfo](#)
- [VkCommandBufferInheritanceInfo](#)
- [VkCommandPoolCreateInfo](#)
- [VkComponentMapping](#)
- [VkComputePipelineCreateInfo](#)
- [VkCopyDescriptorSet](#)
- [VkDescriptorBufferInfo](#)
- [VkDescriptorImageInfo](#)
- [VkDescriptorPoolCreateInfo](#)
- [VkDescriptorPoolSize](#)
- [VkDescriptorSetAllocateInfo](#)
- [VkDescriptorSetLayoutBinding](#)
- [VkDescriptorSetLayoutCreateInfo](#)
- [VkDeviceCreateInfo](#)
- [VkDeviceQueueCreateInfo](#)
- [VkDispatchIndirectCommand](#)
- [VkDrawIndexedIndirectCommand](#)
- [VkDrawIndirectCommand](#)
- [VkEventCreateInfo](#)
- [VkExtensionProperties](#)
- [VkExtent2D](#)
- [VkExtent3D](#)
- [VkFenceCreateInfo](#)
- [VkFormatProperties](#)
- [VkFramebufferCreateInfo](#)
- [VkGraphicsPipelineCreateInfo](#)
- [VkImageBlit](#)
- [VkImageCopy](#)
- [VkImageCreateInfo](#)
- [VkImageFormatProperties](#)

- [VkImageMemoryBarrier](#)
- [VkImageResolve](#)
- [VkImageSubresource](#)
- [VkImageSubresourceLayers](#)
- [VkImageSubresourceRange](#)
- [VkImageViewCreateInfo](#)
- [VkInstanceCreateInfo](#)
- [VkLayerProperties](#)
- [VkMappedMemoryRange](#)
- [VkMemoryAllocateInfo](#)
- [VkMemoryBarrier](#)
- [VkMemoryHeap](#)
- [VkMemoryRequirements](#)
- [VkMemoryType](#)
- [VkOffset2D](#)
- [VkOffset3D](#)
- [VkPhysicalDeviceFeatures](#)
- [VkPhysicalDeviceLimits](#)
- [VkPhysicalDeviceMemoryProperties](#)
- [VkPhysicalDeviceProperties](#)
- [VkPhysicalDeviceSparseProperties](#)
- [VkPipelineCacheCreateInfo](#)
- [VkPipelineCacheHeaderVersionOne](#)
- [VkPipelineColorBlendAttachmentState](#)
- [VkPipelineColorBlendStateCreateInfo](#)
- [VkPipelineDepthStencilStateCreateInfo](#)
- [VkPipelineDynamicStateCreateInfo](#)
- [VkPipelineInputAssemblyStateCreateInfo](#)
- [VkPipelineMultisampleStateCreateInfo](#)
- [VkPipelineRasterizationStateCreateInfo](#)
- [VkPipelineShaderStageCreateInfo](#)
- [VkPipelineTessellationStateCreateInfo](#)
- [VkPipelineVertexInputStateCreateInfo](#)
- [VkPipelineViewportStateCreateInfo](#)
- [VkPushConstantRange](#)

- [VkQueryPoolCreateInfo](#)
- [VkQueueFamilyProperties](#)
- [VkRect2D](#)
- [VkRenderPassBeginInfo](#)
- [VkRenderPassCreateInfo](#)
- [VkSamplerCreateInfo](#)
- [VkSemaphoreCreateInfo](#)
- [VkSparseBufferMemoryBindInfo](#)
- [VkSparseImageFormatProperties](#)
- [VkSparseImageMemoryBind](#)
- [VkSparseImageMemoryBindInfo](#)
- [VkSparseImageMemoryRequirements](#)
- [VkSparseImageOpaqueMemoryBindInfo](#)
- [VkSparseMemoryBind](#)
- [VkSpecializationInfo](#)
- [VkSpecializationMapEntry](#)
- [VkStencilOpState](#)
- [VkSubmitInfo](#)
- [VkSubpassDependency](#)
- [VkSubpassDescription](#)
- [VkSubresourceLayout](#)
- [VkVertexInputAttributeDescription](#)
- [VkVertexInputBindingDescription](#)
- [VkViewport](#)
- [VkWriteDescriptorSet](#)
- Extending [VkBindDescriptorSetsInfo](#), [VkPushConstantsInfo](#), [VkPushDescriptorSetInfo](#), [VkPushDescriptorSetWithTemplateInfo](#), [VkSetDescriptorBufferOffsetsInfoEXT](#), [VkBindDescriptorBufferEmbeddedSamplersInfoEXT](#), [VkIndirectCommandsLayoutCreateInfoEXT](#):
 - [VkPipelineLayoutCreateInfo](#)
- Extending [VkPipelineShaderStageCreateInfo](#), [VkDataGraphPipelineCreateInfoARM](#):
 - [VkShaderModuleCreateInfo](#)

New Unions

- [VkClearColorValue](#)
- [VkClearValue](#)

New Function Pointers

- [PFN_vkAllocationFunction](#)
- [PFN_vkFreeFunction](#)
- [PFN_vkInternalAllocationNotification](#)
- [PFN_vkInternalFreeNotification](#)
- [PFN_vkReallocationFunction](#)
- [PFN_vkVoidFunction](#)

New Enums

- [VkAccessFlagBits](#)
- [VkAttachmentDescriptionFlagBits](#)
- [VkAttachmentLoadOp](#)
- [VkAttachmentStoreOp](#)
- [VkBlendFactor](#)
- [VkBlendOp](#)
- [VkBorderColor](#)
- [VkBufferCreateFlagBits](#)
- [VkBufferUsageFlagBits](#)
- [VkColorComponentFlagBits](#)
- [VkCommandBufferLevel](#)
- [VkCommandBufferResetFlagBits](#)
- [VkCommandBufferUsageFlagBits](#)
- [VkCommandPoolCreateFlagBits](#)
- [VkCommandPoolResetFlagBits](#)
- [VkCompareOp](#)
- [VkComponentSwizzle](#)
- [VkCullModeFlagBits](#)
- [VkDependencyFlagBits](#)
- [VkDescriptorPoolCreateFlagBits](#)
- [VkDescriptorSetLayoutCreateFlagBits](#)
- [VkDescriptorType](#)
- [VkDynamicState](#)
- [VkEventCreateFlagBits](#)
- [VkFenceCreateFlagBits](#)

- [VkFilter](#)
- [VkFormat](#)
- [VkFormatFeatureFlagBits](#)
- [VkFramebufferCreateFlagBits](#)
- [VkFrontFace](#)
- [VkImageAspectFlagBits](#)
- [VkImageCreateFlagBits](#)
- [VkImageLayout](#)
- [VkImageTiling](#)
- [VkImageType](#)
- [VkImageUsageFlagBits](#)
- [VkImageViewCreateFlagBits](#)
- [VkImageViewType](#)
- [VkIndexType](#)
- [VkInstanceCreateFlagBits](#)
- [VkInternalAllocationType](#)
- [VkLogicOp](#)
- [VkMemoryHeapFlagBits](#)
- [VkMemoryMapFlagBits](#)
- [VkMemoryPropertyFlagBits](#)
- [VkObjectType](#)
- [VkPhysicalDeviceType](#)
- [VkPipelineBindPoint](#)
- [VkPipelineCacheHeaderVersion](#)
- [VkPipelineCreateFlagBits](#)
- [VkPipelineShaderStageCreateFlagBits](#)
- [VkPipelineStageFlagBits](#)
- [VkPolygonMode](#)
- [VkPrimitiveTopology](#)
- [VkQueryControlFlagBits](#)
- [VkQueryPipelineStatisticFlagBits](#)
- [VkQueryPoolCreateFlagBits](#)
- [VkQueryResultFlagBits](#)
- [VkQueryType](#)
- [VkQueueFlagBits](#)

- [VkRenderPassCreateFlagBits](#)
- [VkResult](#)
- [VkSampleCountFlagBits](#)
- [VkSamplerAddressMode](#)
- [VkSamplerCreateFlagBits](#)
- [VkSamplerMipmapMode](#)
- [VkShaderStageFlagBits](#)
- [VkSharingMode](#)
- [VkSparseImageFormatFlagBits](#)
- [VkSparseMemoryBindFlagBits](#)
- [VkStencilFaceFlagBits](#)
- [VkStencilOp](#)
- [VkStructureType](#)
- [VkSubpassContents](#)
- [VkSubpassDescriptionFlagBits](#)
- [VkSystemAllocationScope](#)
- [VkVendorId](#)
- [VkVertexInputRate](#)

New Bitmasks

- [VkAccessFlags](#)
- [VkAttachmentDescriptionFlags](#)
- [VkBufferCreateFlags](#)
- [VkBufferUsageFlags](#)
- [VkBufferViewCreateFlags](#)
- [VkColorComponentFlags](#)
- [VkCommandBufferResetFlags](#)
- [VkCommandBufferUsageFlags](#)
- [VkCommandPoolCreateFlags](#)
- [VkCommandPoolResetFlags](#)
- [VkCullModeFlags](#)
- [VkDependencyFlags](#)
- [VkDescriptorPoolCreateFlags](#)
- [VkDescriptorPoolResetFlags](#)
- [VkDescriptorSetLayoutCreateFlags](#)

- [VkDeviceCreateFlags](#)
- [VkDeviceQueueCreateFlags](#)
- [VkEventCreateFlags](#)
- [VkFenceCreateFlags](#)
- [VkFormatFeatureFlags](#)
- [VkFramebufferCreateFlags](#)
- [VkImageAspectFlags](#)
- [VkImageCreateFlags](#)
- [VkImageUsageFlags](#)
- [VkImageViewCreateFlags](#)
- [VkInstanceCreateFlags](#)
- [VkMemoryHeapFlags](#)
- [VkMemoryMapFlags](#)
- [VkMemoryPropertyFlags](#)
- [VkPipelineCacheCreateFlags](#)
- [VkPipelineColorBlendStateCreateFlags](#)
- [VkPipelineCreateFlags](#)
- [VkPipelineDepthStencilStateCreateFlags](#)
- [VkPipelineDynamicStateCreateFlags](#)
- [VkPipelineInputAssemblyStateCreateFlags](#)
- [VkPipelineLayoutCreateFlags](#)
- [VkPipelineMultisampleStateCreateFlags](#)
- [VkPipelineRasterizationStateCreateFlags](#)
- [VkPipelineShaderStageCreateFlags](#)
- [VkPipelineStageFlags](#)
- [VkPipelineTessellationStateCreateFlags](#)
- [VkPipelineVertexInputStateCreateFlags](#)
- [VkPipelineViewportStateCreateFlags](#)
- [VkQueryControlFlags](#)
- [VkQueryPipelineStatisticFlags](#)
- [VkQueryPoolCreateFlags](#)
- [VkQueryResultFlags](#)
- [VkQueueFlags](#)
- [VkRenderPassCreateFlags](#)
- [VkSampleCountFlags](#)

- [VkSamplerCreateFlags](#)
- [VkSemaphoreCreateFlags](#)
- [VkShaderModuleCreateFlags](#)
- [VkShaderStageFlags](#)
- [VkSparseImageFormatFlags](#)
- [VkSparseMemoryBindFlags](#)
- [VkStencilFaceFlags](#)
- [VkSubpassDescriptionFlags](#)

New Headers

- [vk_platform](#)

New Enum Constants

- [VK_ATTACHMENT_UNUSED](#)
- [VK_FALSE](#)
- [VK_LOD_CLAMP_NONE](#)
- [VK_MAX_DESCRIPTION_SIZE](#)
- [VK_MAX_EXTENSION_NAME_SIZE](#)
- [VK_MAX_MEMORY_HEAPS](#)
- [VK_MAX_MEMORY_TYPES](#)
- [VK_MAX_PHYSICAL_DEVICE_NAME_SIZE](#)
- [VK_QUEUE_FAMILY_IGNORED](#)
- [VK_REMAINING_ARRAY_LAYERS](#)
- [VK_REMAINING_MIP_LEVELS](#)
- [VK_SUBPASS_EXTERNAL](#)
- [VK_TRUE](#)
- [VK_UUID_SIZE](#)
- [VK_WHOLE_SIZE](#)
- Extending [VkResult](#):
 - [VK_ERROR_VALIDATION_FAILED](#)

Vulkan SC Version 1.0

Vulkan SC Version 1.0 is a variant of the Vulkan 1.2 API that is tailored for safety-critical use cases.

New Macros

- [VKSC_API_VARIANT](#)
- [VKSC_API_VERSION_1_0](#)

New Commands

- [vkGetCommandPoolMemoryConsumption](#)
- [vkGetFaultData](#)

New Structures

- [VkCommandPoolMemoryConsumption](#)
- [VkFaultData](#)
- [VkPipelineCacheHeaderVersionSafetyCriticalOne](#)
- [VkPipelineCacheSafetyCriticalIndexEntry](#)
- [VkPipelineCacheStageValidationIndexEntry](#)
- [VkPipelinePoolSize](#)
- Extending [VkCommandPoolCreateInfo](#):
 - [VkCommandPoolMemoryReservationCreateInfo](#)
- Extending [VkDeviceCreateInfo](#):
 - [VkDeviceObjectReservationCreateInfo](#)
 - [VkFaultCallbackInfo](#)
- Extending [VkGraphicsPipelineCreateInfo](#), [VkComputePipelineCreateInfo](#), [VkRayTracingPipelineCreateInfoKHR](#), [VkRayTracingPipelineCreateInfoNV](#):
 - [VkPipelineOfflineCreateInfo](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceVulkanSC10Features](#)
- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDeviceVulkanSC10Properties](#)

New Function Pointers

- [PFN_vkFaultCallbackFunction](#)

New Enums

- [VkFaultLevel](#)
- [VkFaultQueryBehavior](#)
- [VkFaultType](#)

- [VkPipelineCacheCreateFlagBits](#)
- [VkPipelineCacheValidationVersion](#)
- [VkPipelineMatchControl](#)

New Enum Constants

- Extending [VkMemoryHeapFlagBits](#):
 - `VK_MEMORY_HEAP_SEU_SAFE_BIT`
- Extending [VkPipelineCacheCreateFlagBits](#):
 - `VK_PIPELINE_CACHE_CREATE_READ_ONLY_BIT`
 - `VK_PIPELINE_CACHE_CREATE_USE_APPLICATION_STORAGE_BIT`
- Extending [VkPipelineCacheHeaderVersion](#):
 - `VK_PIPELINE_CACHE_HEADER_VERSION_SAFETY_CRITICAL_ONE`
- Extending [VkResult](#):
 - `VK_ERROR_INVALID_PIPELINE_CACHE_DATA`
 - `VK_ERROR_NO_PIPELINE_MATCH`
- Extending [VkStructureType](#):
 - `VK_STRUCTURE_TYPE_COMMAND_POOL_MEMORY_CONSUMPTION`
 - `VK_STRUCTURE_TYPE_COMMAND_POOL_MEMORY_RESERVATION_CREATE_INFO`
 - `VK_STRUCTURE_TYPE_DEVICE_OBJECT_RESERVATION_CREATE_INFO`
 - `VK_STRUCTURE_TYPE_FAULT_CALLBACK_INFO`
 - `VK_STRUCTURE_TYPE_FAULT_DATA`
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_SC_1_0_FEATURES`
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_SC_1_0_PROPERTIES`
 - `VK_STRUCTURE_TYPE_PIPELINE_OFFLINE_CREATE_INFO`
 - `VK_STRUCTURE_TYPE_PIPELINE_POOL_SIZE`

Appendix E: Layers & Extensions

(Informative)

Extensions to the Vulkan API **can** be defined by authors, groups of authors, and the Khronos Vulkan Safety Critical Working Group. The online Registry of extensions is available at URL

<https://registry.khronos.org/vulkansc/>

Authors creating extensions and layers **must** follow the mandatory procedures described in the [Vulkan Documentation and Extensions](#) document when creating extensions and layers.

The version of the Vulkan specification published in the Registry includes all registered and published extensions. It is also possible to build the specification with an arbitrary set of extensions included, and the specification you are viewing may not include all extensions.

In the remainder of this appendix, extensions are grouped as Khronos **KHR**, multivendor **EXT**, and then alphabetically by author ID. Within each group, extensions are listed in alphabetical order by their name.

Extension Dependencies

Extensions which have dependencies on specific core versions or on other extensions will list such dependencies.

For core versions, the specified version **must** be supported at runtime. All extensions implicitly require support for Vulkan 1.0.

For a device extension, use of any device-level functionality defined by that extension requires that any extensions that extension depends on be enabled.

For any extension, use of any instance-level functionality defined by that extension requires only that any extensions that extension depends on be supported at runtime.

Extension Interactions

Some extensions define APIs which are only supported when other extensions or core versions are supported at runtime. Such interactions are noted as “API Interactions”.

List of Current Extensions

- [VK_KHR_calibrated_timestamps](#)
- [VK_KHR_copy_commands2](#) (promoted to core)
- [VK_KHR_display](#)
- [VK_KHR_display_swapchain](#)
- [VK_KHR_external_fence_fd](#)

- [VK_KHR_external_memory_fd](#)
- [VK_KHR_external_semaphore_fd](#)
- [VK_KHR_fragment_shading_rate](#)
- [VK_KHR_get_display_properties2](#)
- [VK_KHR_get_surface_capabilities2](#)
- [VK_KHR_global_priority](#) (promoted to core)
- [VK_KHR_incremental_present](#)
- [VK_KHR_index_type_uint8](#) (promoted to core)
- [VK_KHR_line_rasterization](#) (promoted to core)
- [VK_KHR_object_refresh](#)
- [VK_KHR_performance_query](#)
- [VK_KHR_shader_clock](#)
- [VK_KHR_shader_terminate_invocation](#) (promoted to core)
- [VK_KHR_shared_presentable_image](#)
- [VK_KHR_surface](#)
- [VK_KHR_swapchain](#)
- [VK_KHR_swapchain_mutable_format](#)
- [VK_KHR_synchronization2](#) (promoted to core)
- [VK_KHR_vertex_attribute_divisor](#) (promoted to core)
- [VK_EXT_4444_formats](#) (promoted to core)
- [VK_EXT_application_parameters](#)
- [VK_EXT_astc_decode_mode](#)
- [VK_EXT_blend_operation_advanced](#)
- [VK_EXT_color_write_enable](#)
- [VK_EXT_conservative_rasterization](#)
- [VK_EXT_custom_border_color](#)
- [VK_EXT_debug_utils](#)
- [VK_EXT_depth_clip_enable](#)
- [VK_EXT_depth_range_unrestricted](#)
- [VK_EXT_direct_mode_display](#)
- [VK_EXT_discard_rectangles](#)
- [VK_EXT_display_control](#)
- [VK_EXT_display_surface_counter](#)
- [VK_EXT_extended_dynamic_state](#) (promoted to core)
- [VK_EXT_extended_dynamic_state2](#) (promoted to core)

- [VK_EXT_external_memory_dma_buf](#)
- [VK_EXT_external_memory_host](#)
- [VK_EXT_filter_cubic](#)
- [VK_EXT_fragment_shader_interlock](#)
- [VK_EXT_hdr_metadata](#)
- [VK_EXT_headless_surface](#)
- [VK_EXT_image_drm_format_modifier](#)
- [VK_EXT_image_robustness](#) (promoted to core)
- [VK_EXT_layer_settings](#)
- [VK_EXT_memory_budget](#)
- [VK_EXT_pci_bus_info](#)
- [VK_EXT_post_depth_coverage](#)
- [VK_EXT_queue_family_foreign](#)
- [VK_EXT_robustness2](#)
- [VK_EXT_sample_locations](#)
- [VK_EXT_shader_atomic_float](#)
- [VK_EXT_shader_demote_to_helper_invocation](#) (promoted to core)
- [VK_EXT_shader_image_atomic_int64](#)
- [VK_EXT_shader_stencil_export](#)
- [VK_EXT_subgroup_size_control](#) (promoted to core)
- [VK_EXT_swapchain_colorspace](#)
- [VK_EXT_texel_buffer_alignment](#) (promoted to core)
- [VK_EXT_texture_compression_astc_hdr](#) (promoted to core)
- [VK_EXT_vertex_input_dynamic_state](#)
- [VK_EXT_ycbcr_2plane_444_formats](#) (promoted to core)
- [VK_EXT_ycbcr_image_arrays](#)
- [VK_NV_acquire_winrt_display](#)
- [VK_NV_external_memory_sci_buf](#)
- [VK_NV_external_sci_sync2](#)
- [VK_NV_private_vendor_info](#)
- [VK_QNX_external_memory_screen_buffer](#)

VK_KHR_calibrated_timestamps

Name String

VK_KHR_calibrated_timestamps

Extension Type

Device extension

Registered Extension Number

544

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

[Vulkan Version 1.1](#)

Contact

- Daniel Rakos [\[GitHub\]](#)[aqnuep](#)

Other Extension Metadata

Last Modified Date

2023-07-12

IP Status

No known IP claims.

Contributors

- Matthaeus G. Chajdas, AMD
- Alan Harrison, AMD
- Derrick Owens, AMD
- Daniel Rakos, RasterGrid
- Faith Ekstrand, Intel
- Keith Packard, Valve

Description

This extension provides an interface to query calibrated timestamps obtained quasi simultaneously from two time domains.

New Commands

- [vkGetCalibratedTimestampsKHR](#)
- [vkGetPhysicalDeviceCalibrateableTimeDomainsKHR](#)

New Structures

- [VkCalibratedTimestampInfoKHR](#)

New Enums

- [VkTimeDomainKHR](#)

New Enum Constants

- [VK_KHR_CALIBRATED_TIMESTAMPS_EXTENSION_NAME](#)
- [VK_KHR_CALIBRATED_TIMESTAMPS_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_CALIBRATED_TIMESTAMP_INFO_KHR](#)

Version History

- Revision 1, 2023-07-12 (Daniel Rakos)
 - Initial draft.

VK_KHR_copy_commands2

Name String

[VK_KHR_copy_commands2](#)

Extension Type

Device extension

Registered Extension Number

338

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_get_physical_device_properties2](#)

or

[Vulkan Version 1.1](#)

Deprecation State

- *Promoted* to Vulkan 1.3

Contact

- Matthew Netsch [\[GitHub\]](#)[mnetsch](#)

Other Extension Metadata

Last Modified Date

2020-07-06

Contributors

- Jeff Leger, Qualcomm
- Tobias Hector, AMD
- Jan-Harald Fredriksen, ARM
- Tom Olson, ARM

Description

This extension provides extensible versions of the Vulkan buffer and image copy commands. The new commands are functionally identical to the core commands, except that their copy parameters are specified using extensible structures that can be used to pass extension-specific information.

The following extensible copy commands are introduced with this extension: [vkCmdCopyBuffer2KHR](#), [vkCmdCopyImage2KHR](#), [vkCmdCopyBufferToImage2KHR](#), [vkCmdCopyImageToBuffer2KHR](#), [vkCmdBlitImage2KHR](#), and [vkCmdResolveImage2KHR](#). Each command contains an **Info2KHR* structure parameter that includes *sType/pNext* members. Lower level structures describing each region to be copied are also extended with *sType/pNext* members.

New Commands

- [vkCmdBlitImage2KHR](#)
- [vkCmdCopyBuffer2KHR](#)
- [vkCmdCopyBufferToImage2KHR](#)
- [vkCmdCopyImage2KHR](#)
- [vkCmdCopyImageToBuffer2KHR](#)
- [vkCmdResolveImage2KHR](#)

New Structures

- [VkBlitImageInfo2KHR](#)
- [VkBufferCopy2KHR](#)
- [VkBufferImageCopy2KHR](#)
- [VkCopyBufferInfo2KHR](#)
- [VkCopyBufferToImageInfo2KHR](#)

- [VkCopyImageInfo2KHR](#)
- [VkCopyImageToBufferInfo2KHR](#)
- [VkImageBlit2KHR](#)
- [VkImageCopy2KHR](#)
- [VkImageResolve2KHR](#)
- [VkResolveImageInfo2KHR](#)

New Enum Constants

- `VK_KHR_COPY_COMMANDS_2_EXTENSION_NAME`
- `VK_KHR_COPY_COMMANDS_2_SPEC_VERSION`
- Extending [VkStructureType](#):
 - `VK_STRUCTURE_TYPE_BLIT_IMAGE_INFO_2_KHR`
 - `VK_STRUCTURE_TYPE_BUFFER_COPY_2_KHR`
 - `VK_STRUCTURE_TYPE_BUFFER_IMAGE_COPY_2_KHR`
 - `VK_STRUCTURE_TYPE_COPY_BUFFER_INFO_2_KHR`
 - `VK_STRUCTURE_TYPE_COPY_BUFFER_TO_IMAGE_INFO_2_KHR`
 - `VK_STRUCTURE_TYPE_COPY_IMAGE_INFO_2_KHR`
 - `VK_STRUCTURE_TYPE_COPY_IMAGE_TO_BUFFER_INFO_2_KHR`
 - `VK_STRUCTURE_TYPE_IMAGE_BLIT_2_KHR`
 - `VK_STRUCTURE_TYPE_IMAGE_COPY_2_KHR`
 - `VK_STRUCTURE_TYPE_IMAGE_RESOLVE_2_KHR`
 - `VK_STRUCTURE_TYPE_RESOLVE_IMAGE_INFO_2_KHR`

Promotion to Vulkan 1.3

Vulkan APIs in this extension are included in core Vulkan 1.3, with the KHR suffix omitted. External interactions defined by this extension, such as SPIR-V token names, retain their original names. The original Vulkan API names are still available as aliases of the core functionality.

Version History

- Revision 1, 2020-07-06 (Jeff Leger)
 - Internal revisions

VK_KHR_display

Name String

`VK_KHR_display`

Extension Type

Instance extension

Registered Extension Number

3

Revision

23

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_surface](#)

Contact

- James Jones [\[GitHub\]](#)[cubanismo](#)
- Norbert Nopper [\[GitHub\]](#)[FslNopper](#)

Other Extension Metadata

Last Modified Date

2017-03-13

IP Status

No known IP claims.

Contributors

- James Jones, NVIDIA
- Norbert Nopper, Freescale
- Jeff Vigil, Qualcomm
- Daniel Rakos, AMD

Description

This extension provides the API to enumerate displays and available modes on a given device.

New Object Types

- [VkDisplayKHR](#)
- [VkDisplayModeKHR](#)

New Commands

- [vkCreateDisplayModeKHR](#)
- [vkCreateDisplayPlaneSurfaceKHR](#)

- [vkGetDisplayModePropertiesKHR](#)
- [vkGetDisplayPlaneCapabilitiesKHR](#)
- [vkGetDisplayPlaneSupportedDisplaysKHR](#)
- [vkGetPhysicalDeviceDisplayPlanePropertiesKHR](#)
- [vkGetPhysicalDeviceDisplayPropertiesKHR](#)

New Structures

- [VkDisplayModeCreateInfoKHR](#)
- [VkDisplayModeParametersKHR](#)
- [VkDisplayModePropertiesKHR](#)
- [VkDisplayPlaneCapabilitiesKHR](#)
- [VkDisplayPlanePropertiesKHR](#)
- [VkDisplayPropertiesKHR](#)
- [VkDisplaySurfaceCreateInfoKHR](#)

New Enums

- [VkDisplayPlaneAlphaFlagBitsKHR](#)

New Bitmasks

- [VkDisplayModeCreateFlagsKHR](#)
- [VkDisplayPlaneAlphaFlagsKHR](#)
- [VkDisplaySurfaceCreateFlagsKHR](#)
- [VkSurfaceTransformFlagsKHR](#)

New Enum Constants

- [VK_KHR_DISPLAY_EXTENSION_NAME](#)
- [VK_KHR_DISPLAY_SPEC_VERSION](#)
- Extending [VkObjectType](#):
 - [VK_OBJECT_TYPE_DISPLAY_KHR](#)
 - [VK_OBJECT_TYPE_DISPLAY_MODE_KHR](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_DISPLAY_MODE_CREATE_INFO_KHR](#)
 - [VK_STRUCTURE_TYPE_DISPLAY_SURFACE_CREATE_INFO_KHR](#)

Issues

1) Which properties of a mode should be fixed in the mode information vs. settable in some other function when setting the mode? E.g., do we need to double the size of the mode pool to include

both stereo and non-stereo modes? YUV and RGB scanout even if they both take RGB input images? BGR vs. RGB input? etc.

RESOLVED: Many modern displays support at most a handful of resolutions and timings natively. Other “modes” are expected to be supported using scaling hardware on the display engine or GPU. Other properties, such as rotation and mirroring should not require duplicating hardware modes just to express all combinations. Further, these properties may be implemented on a per-display or per-overlay granularity.

To avoid the exponential growth of modes as mutable properties are added, as was the case with `EGLConfig`/WGL pixel formats/`GLXFBConfig`, this specification should separate out hardware properties and configurable state into separate objects. Modes and overlay planes will express capabilities of the hardware, while a separate structure will allow applications to configure scaling, rotation, mirroring, color keys, LUT values, alpha masks, etc. for a given swapchain independent of the mode in use. Constraints on these settings will be established by properties of the immutable objects.

Note the resolution of this issue may affect issue 5 as well.

2) What properties of a display itself are useful?

RESOLVED: This issue is too broad. It was meant to prompt general discussion, but resolving this issue amounts to completing this specification. All interesting properties should be included. The issue will remain as a placeholder since removing it would make it hard to parse existing discussion notes that refer to issues by number.

3) How are multiple overlay planes within a display or mode enumerated?

RESOLVED: They are referred to by an index. Each display will report the number of overlay planes it contains.

4) Should swapchains be created relative to a mode or a display?

RESOLVED: When using this extension, swapchains are created relative to a mode and a plane. The mode implies the display object the swapchain will present to. If the specified mode is not the display’s current mode, the new mode will be applied when the first image is presented to the swapchain, and the default operating system mode, if any, will be restored when the swapchain is destroyed.

5) Should users query generic ranges from displays and construct their own modes explicitly using those constraints rather than querying a fixed set of modes (Most monitors only have one real “mode” these days, even though many support relatively arbitrary scaling, either on the monitor side or in the GPU display engine, making “modes” something of a relic/compatibility construct).

RESOLVED: Expose both. Display information structures will expose a set of predefined modes, as well as any attributes necessary to construct a customized mode.

6) Is it fine if we return the display and display mode handles in the structure used to query their properties?

RESOLVED: Yes.

7) Is there a possibility that not all displays of a device work with all of the present queues of a device? If yes, how do we determine which displays work with which present queues?

RESOLVED: No known hardware has such limitations, but determining such limitations is supported automatically using the existing `VK_KHR_surface` and `VK_KHR_swapchain` query mechanisms.

8) Should all presentation need to be done relative to an overlay plane, or can a display mode + display be used alone to target an output?

RESOLVED: Require specifying a plane explicitly.

9) Should displays have an associated window system display, such as an `HDC` or `Display*`?

RESOLVED: No. Displays are independent of any windowing system in use on the system. Further, neither `HDC` nor `Display*` refer to a physical display object.

10) Are displays queried from a physical GPU or from a device instance?

RESOLVED: Developers prefer to query modes directly from the physical GPU so they can use display information as an input to their device selection algorithms prior to device creation. This avoids the need to create placeholder device instances to enumerate displays.

This preference must be weighed against the extra initialization that must be done by driver vendors prior to device instance creation to support this usage.

11) Should displays and/or modes be dispatchable objects? If functions are to take displays, overlays, or modes as their first parameter, they must be dispatchable objects as defined in Khronos bug 13529. If they are not added to the list of dispatchable objects, functions operating on them must take some higher-level object as their first parameter. There is no performance case against making them dispatchable objects, but they would be the first extension objects to be dispatchable.

RESOLVED: Do not make displays or modes dispatchable. They will dispatch based on their associated physical device.

12) Should hardware cursor capabilities be exposed?

RESOLVED: Defer. This could be a separate extension on top of the base WSI specs.

13) How many display objects should be enumerated for "tiled" display devices? There are ongoing design discussions among lower-level display API authors regarding how to expose displays if they are one physical display device to an end user, but may internally be implemented as two side-by-side displays using the same display engine (and sometimes cabling) resources as two physically separate display devices.

RESOLVED: Tiled displays will appear as a single display object in this API.

14) Should the raw EDID data be included in the display information?

RESOLVED: No. A future extension could be added which reports the EDID if necessary. This may be complicated by the outcome of issue 13.

15) Should min and max scaling factor capabilities of overlays be exposed?

RESOLVED: Yes. This is exposed indirectly by allowing applications to query the min/max position and extent of the source and destination regions from which image contents are fetched by the display engine when using a particular mode and overlay pair.

16) Should devices be able to expose planes that can be moved between displays? If so, how?

RESOLVED: Yes. Applications can determine which displays a given plane supports using [vkGetDisplayPlaneSupportedDisplaysKHR](#).

17) Should there be a way to destroy display modes? If so, does it support destroying “built in” modes?

RESOLVED: Not in this extension. A future extension could add this functionality.

18) What should the lifetime of display and built-in display mode objects be?

RESOLVED: The lifetime of the instance. These objects cannot be destroyed. A future extension may be added to expose a way to destroy these objects and/or support display hotplug.

19) Should persistent mode for smart panels be enabled/disabled at swapchain creation time, or on a per-present basis.

RESOLVED: On a per-present basis.

Version History

- Revision 1, 2015-02-24 (James Jones)
 - Initial draft
- Revision 2, 2015-03-12 (Norbert Nopper)
 - Added overlay enumeration for a display.
- Revision 3, 2015-03-17 (Norbert Nopper)
 - Fixed typos and namings as discussed in Bugzilla.
 - Reordered and grouped functions.
 - Added functions to query count of display, mode and overlay.
 - Added native display handle, which may be needed on some platforms to create a native Window.
- Revision 4, 2015-03-18 (Norbert Nopper)
 - Removed primary and virtualPosition members (see comment of James Jones in Bugzilla).
 - Added native overlay handle to information structure.
 - Replaced , with ; in struct.
- Revision 6, 2015-03-18 (Daniel Rakos)
 - Added WSI extension suffix to all items.

- Made the whole API more “Vulkanish”.
- Replaced all functions with a single `vkGetDisplayInfoKHR` function to better match the rest of the API.
- Made the display, display mode, and overlay objects be first class objects, not subclasses of `VkBaseObject` as they do not support the common functions anyways.
- Renamed `*Info` structures to `*Properties`.
- Removed `overlayIndex` field from `VkOverlayProperties` as there is an implicit index already as a result of moving to a “Vulkanish” API.
- Displays are not get through device, but through physical GPU to match the rest of the Vulkan API. Also this is something ISVs explicitly requested.
- Added issue (6) and (7).
- Revision 7, 2015-03-25 (James Jones)
 - Added an issues section
 - Added rotation and mirroring flags
- Revision 8, 2015-03-25 (James Jones)
 - Combined the duplicate issues sections introduced in last change.
 - Added proposed resolutions to several issues.
- Revision 9, 2015-04-01 (Daniel Rakos)
 - Rebased extension against Vulkan 0.82.0
- Revision 10, 2015-04-01 (James Jones)
 - Added issues (10) and (11).
 - Added more straw-man issue resolutions, and cleaned up the proposed resolution for issue (4).
 - Updated the rotation and mirroring enums to have proper bitmask semantics.
- Revision 11, 2015-04-15 (James Jones)
 - Added proposed resolution for issues (1) and (2).
 - Added issues (12), (13), (14), and (15)
 - Removed `pNativeHandle` field from overlay structure.
 - Fixed small compilation errors in example code.
- Revision 12, 2015-07-29 (James Jones)
 - Rewrote the guts of the extension against the latest WSI swapchain specifications and the latest Vulkan API.
 - Address overlay planes by their index rather than an object handle and refer to them as “planes” rather than “overlays” to make it slightly clearer that even a display with no “overlays” still has at least one base “plane” that images can be displayed on.
 - Updated most of the issues.
 - Added an “extension type” section to the specification header.

- Reused the `VK_EXT_KHR_surface` surface transform enumerations rather than redefining them here.
- Updated the example code to use the new semantics.
- Revision 13, 2015-08-21 (Ian Elliott)
 - Renamed this extension and all of its enumerations, types, functions, etc. This makes it compliant with the proposed standard for Vulkan extensions.
 - Switched from “revision” to “version”, including use of the `VK_MAKE_VERSION` macro in the header file.
- Revision 14, 2015-09-01 (James Jones)
 - Restore single-field revision number.
- Revision 15, 2015-09-08 (James Jones)
 - Added alpha flags enum.
 - Added premultiplied alpha support.
- Revision 16, 2015-09-08 (James Jones)
 - Added description section to the spec.
 - Added issues 16 - 18.
- Revision 17, 2015-10-02 (James Jones)
 - Planes are now a property of the entire device rather than individual displays. This allows planes to be moved between multiple displays on devices that support it.
 - Added a function to create a `VkSurfaceKHR` object describing a display plane and mode to align with the new per-platform surface creation conventions.
 - Removed detailed mode timing data. It was agreed that the mode extents and refresh rate are sufficient for current use cases. Other information could be added back in as an extension if it is needed in the future.
 - Added support for smart/persistent/buffered display devices.
- Revision 18, 2015-10-26 (Ian Elliott)
 - Renamed from `VK_EXT_KHR_display` to `VK_KHR_display`.
- Revision 19, 2015-11-02 (James Jones)
 - Updated example code to match revision 17 changes.
- Revision 20, 2015-11-03 (Daniel Rakos)
 - Added allocation callbacks to creation functions.
- Revision 21, 2015-11-10 (Jesse Hall)
 - Added `VK_DISPLAY_PLANE_ALPHA_OPAQUE_BIT_KHR`, and use `VkDisplayPlaneAlphaFlagBitsKHR` for `VkDisplayPlanePropertiesKHR::alphaMode` instead of `VkDisplayPlaneAlphaFlagsKHR`, since it only represents one mode.
 - Added reserved flags bitmask to `VkDisplayPlanePropertiesKHR`.
 - Use `VkSurfaceTransformFlagBitsKHR` instead of obsolete `VkSurfaceTransformKHR`.

- Renamed `vkGetDisplayPlaneSupportedDisplaysKHR` parameters for clarity.
- Revision 22, 2015-12-18 (James Jones)
 - Added missing “planeIndex” parameter to `vkGetDisplayPlaneSupportedDisplaysKHR()`
- Revision 23, 2017-03-13 (James Jones)
 - Closed all remaining issues. The specification and implementations have been shipping with the proposed resolutions for some time now.
 - Removed the sample code and noted it has been integrated into the official Vulkan SDK cube demo.

VK_KHR_display_swapchain

Name String

VK_KHR_display_swapchain

Extension Type

Device extension

Registered Extension Number

4

Revision

10

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_swapchain](#)

and

[VK_KHR_display](#)

Contact

- James Jones [\[GitHub\]](#)[cubanismo](#)

Other Extension Metadata

Last Modified Date

2017-03-13

IP Status

No known IP claims.

Contributors

- James Jones, NVIDIA
- Jeff Vigil, Qualcomm

- Jesse Hall, Google

Description

This extension provides an API to create a swapchain directly on a device's display without any underlying window system.

New Commands

- [vkCreateSharedSwapchainsKHR](#)

New Structures

- Extending [VkPresentInfoKHR](#):
 - [VkDisplayPresentInfoKHR](#)

New Enum Constants

- [VK_KHR_DISPLAY_SWAPCHAIN_EXTENSION_NAME](#)
- [VK_KHR_DISPLAY_SWAPCHAIN_SPEC_VERSION](#)
- Extending [VkResult](#):
 - [VK_ERROR_INCOMPATIBLE_DISPLAY_KHR](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_DISPLAY_PRESENT_INFO_KHR](#)

Issues

1) Should swapchains sharing images each hold a reference to the images, or should it be up to the application to destroy the swapchains and images in an order that avoids the need for reference counting?

RESOLVED: Take a reference. The lifetime of presentable images is already complex enough.

2) Should the [srcRect](#) and [dstRect](#) parameters be specified as part of the presentation command, or at swapchain creation time?

RESOLVED: As part of the presentation command. This allows moving and scaling the image on the screen without the need to respecify the mode or create a new swapchain and presentable images.

3) Should [srcRect](#) and [dstRect](#) be specified as rects, or separate offset/extent values?

RESOLVED: As rects. Specifying them separately might make it easier for hardware to expose support for one but not the other, but in such cases applications must just take care to obey the reported capabilities and not use non-zero offsets or extents that require scaling, as appropriate.

4) How can applications create multiple swapchains that use the same images?

RESOLVED: By calling [vkCreateSharedSwapchainsKHR](#).

An earlier resolution used `vkCreateSwapchainKHR`, chaining multiple `VkSwapchainCreateInfoKHR` structures through `pNext`. In order to allow each swapchain to also allow other extension structs, a level of indirection was used: `VkSwapchainCreateInfoKHR::pNext` pointed to a different structure, which had both `sType` and `pNext` members for additional extensions, and also had a pointer to the next `VkSwapchainCreateInfoKHR` structure. The number of swapchains to be created could only be found by walking this linked list of alternating structures, and the `pSwapchains` out parameter was reinterpreted to be an array of `VkSwapchainKHR` handles.

Another option considered was a method to specify a “shared” swapchain when creating a new swapchain, such that groups of swapchains using the same images could be built up one at a time. This was deemed unusable because drivers need to know all of the displays an image will be used on when determining which internal formats and layouts to use for that image.

Version History

- Revision 1, 2015-07-29 (James Jones)
 - Initial draft
- Revision 2, 2015-08-21 (Ian Elliott)
 - Renamed this extension and all of its enumerations, types, functions, etc. This makes it compliant with the proposed standard for Vulkan extensions.
 - Switched from “revision” to “version”, including use of the `VK_MAKE_VERSION` macro in the header file.
- Revision 3, 2015-09-01 (James Jones)
 - Restore single-field revision number.
- Revision 4, 2015-09-08 (James Jones)
 - Allow creating multiple swapchains that share the same images using a single call to `vkCreateSwapchainKHR`.
- Revision 5, 2015-09-10 (Alon Or-bach)
 - Removed underscores from `SWAP_CHAIN` in two enums.
- Revision 6, 2015-10-02 (James Jones)
 - Added support for smart panels/buffered displays.
- Revision 7, 2015-10-26 (Ian Elliott)
 - Renamed from `VK_EXT_KHR_display_swapchain` to `VK_KHR_display_swapchain`.
- Revision 8, 2015-11-03 (Daniel Rakos)
 - Updated sample code based on the changes to `VK_KHR_swapchain`.
- Revision 9, 2015-11-10 (Jesse Hall)
 - Replaced `VkDisplaySwapchainCreateInfoKHR` with `vkCreateSharedSwapchainsKHR`, changing resolution of issue #4.
- Revision 10, 2017-03-13 (James Jones)
 - Closed all remaining issues. The specification and implementations have been shipping with the proposed resolutions for some time now.

- Removed the sample code and noted it has been integrated into the official Vulkan SDK cube demo.

VK_KHR_external_fence_fd

Name String

VK_KHR_external_fence_fd

Extension Type

Device extension

Registered Extension Number

116

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_external_fence

or

[Vulkan Version 1.1](#)

Contact

- Jesse Hall [\[GitHub\]](#)[critsec](#)

Other Extension Metadata

Last Modified Date

2017-05-08

IP Status

No known IP claims.

Contributors

- Jesse Hall, Google
- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Cass Everitt, Oculus
- Contributors to [VK_KHR_external_semaphore_fd](#)

Description

An application using external memory may wish to synchronize access to that memory using fences. This extension enables an application to export fence payload to and import fence payload

from POSIX file descriptors.

New Commands

- [vkGetFenceFdKHR](#)
- [vkImportFenceFdKHR](#)

New Structures

- [VkFenceGetFdInfoKHR](#)
- [VkImportFenceFdInfoKHR](#)

New Enum Constants

- [VK_KHR_EXTERNAL_FENCE_FD_EXTENSION_NAME](#)
- [VK_KHR_EXTERNAL_FENCE_FD_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_FENCE_GET_FD_INFO_KHR](#)
 - [VK_STRUCTURE_TYPE_IMPORT_FENCE_FD_INFO_KHR](#)

Issues

This extension borrows concepts, semantics, and language from [VK_KHR_external_semaphore_fd](#). That extension's issues apply equally to this extension.

Version History

- Revision 1, 2017-05-08 (Jesse Hall)
 - Initial revision

VK_KHR_external_memory_fd

Name String

[VK_KHR_external_memory_fd](#)

Extension Type

Device extension

Registered Extension Number

75

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_external_memory](#)

or

[Vulkan Version 1.1](#)

Contact

- James Jones [\[GitHub\]cubanismo](#)

Other Extension Metadata

Last Modified Date

2016-10-21

IP Status

No known IP claims.

Contributors

- James Jones, NVIDIA
- Jeff Juliano, NVIDIA

Description

An application may wish to reference device memory in multiple Vulkan logical devices or instances, in multiple processes, and/or in multiple APIs. This extension enables an application to export POSIX file descriptor handles from Vulkan memory objects and to import Vulkan memory objects from POSIX file descriptor handles exported from other Vulkan memory objects or from similar resources in other APIs.

New Commands

- [vkGetMemoryFdKHR](#)
- [vkGetMemoryFdPropertiesKHR](#)

New Structures

- [VkMemoryFdPropertiesKHR](#)
- [VkMemoryGetFdInfoKHR](#)
- Extending [VkMemoryAllocateInfo](#):
 - [VkImportMemoryFdInfoKHR](#)

New Enum Constants

- [VK_KHR_EXTERNAL_MEMORY_FD_EXTENSION_NAME](#)
- [VK_KHR_EXTERNAL_MEMORY_FD_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_IMPORT_MEMORY_FD_INFO_KHR](#)

- `VK_STRUCTURE_TYPE_MEMORY_FD_PROPERTIES_KHR`
- `VK_STRUCTURE_TYPE_MEMORY_GET_FD_INFO_KHR`

Issues

1) Does the application need to close the file descriptor returned by [vkGetMemoryFdKHR](#)?

RESOLVED: Yes, unless it is passed back in to a driver instance to import the memory. A successful get call transfers ownership of the file descriptor to the application, and a successful import transfers it back to the driver. Destroying the original memory object will not close the file descriptor or remove its reference to the underlying memory resource associated with it.

2) Do drivers ever need to expose multiple file descriptors per memory object?

RESOLVED: No. This would indicate there are actually multiple memory objects, rather than a single memory object.

3) How should the valid size and memory type for POSIX file descriptor memory handles created outside of Vulkan be specified?

RESOLVED: The valid memory types are queried directly from the external handle. The size will be specified by future extensions that introduce such external memory handle types.

Version History

- Revision 1, 2016-10-21 (James Jones)
 - Initial revision

VK_KHR_external_semaphore_fd

Name String

`VK_KHR_external_semaphore_fd`

Extension Type

Device extension

Registered Extension Number

80

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_external_semaphore`

or

[Vulkan Version 1.1](#)

Contact

- James Jones [\[GitHub\]cubanismo](#)

Other Extension Metadata

Last Modified Date

2016-10-21

IP Status

No known IP claims.

Contributors

- Jesse Hall, Google
- James Jones, NVIDIA
- Jeff Juliano, NVIDIA
- Carsten Rohde, NVIDIA

Description

An application using external memory may wish to synchronize access to that memory using semaphores. This extension enables an application to export semaphore payload to and import semaphore payload from POSIX file descriptors.

New Commands

- [vkGetSemaphoreFdKHR](#)
- [vkImportSemaphoreFdKHR](#)

New Structures

- [VkImportSemaphoreFdInfoKHR](#)
- [VkSemaphoreGetFdInfoKHR](#)

New Enum Constants

- `VK_KHR_EXTERNAL_SEMAPHORE_FD_EXTENSION_NAME`
- `VK_KHR_EXTERNAL_SEMAPHORE_FD_SPEC_VERSION`
- Extending [VkStructureType](#):
 - `VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_FD_INFO_KHR`
 - `VK_STRUCTURE_TYPE_SEMAPHORE_GET_FD_INFO_KHR`

Issues

1) Does the application need to close the file descriptor returned by [vkGetSemaphoreFdKHR](#)?

RESOLVED: Yes, unless it is passed back in to a driver instance to import the semaphore. A

successful get call transfers ownership of the file descriptor to the application, and a successful import transfers it back to the driver. Destroying the original semaphore object will not close the file descriptor or remove its reference to the underlying semaphore resource associated with it.

Version History

- Revision 1, 2016-10-21 (Jesse Hall)
 - Initial revision

VK_KHR_fragment_shading_rate

Name String

VK_KHR_fragment_shading_rate

Extension Type

Device extension

Registered Extension Number

227

Revision

2

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

Vulkan Version 1.1

and

VK_KHR_create_renderpass2

or

Vulkan Version 1.2

API Interactions

- Interacts with VK_VERSION_1_3
- Interacts with VK_KHR_dynamic_rendering
- Interacts with VK_KHR_format_feature_flags2

SPIR-V Dependencies

- [SPV_KHR_fragment_shading_rate](#)

Contact

- Tobias Hector [\[GitHub\]](#)tobski

Extension Proposal

[VK_KHR_fragment_shading_rate](#)

Other Extension Metadata

Last Modified Date

2021-09-30

Interactions and External Dependencies

- This extension provides API support for [GL_EXT_fragment_shading_rate](#)

Contributors

- Tobias Hector, AMD
- Guennadi Riguer, AMD
- Matthaeus Chajdas, AMD
- Pat Brown, Nvidia
- Matthew Netsch, Qualcomm
- Slawomir Grajewski, Intel
- Jan-Harald Fredriksen, Arm
- Jeff Bolz, Nvidia
- Arseny Kapoulkine, Roblox
- Contributors to the VK_NV_shading_rate_image specification
- Contributors to the VK_EXT_fragment_density_map specification

Description

This extension adds the ability to change the rate at which fragments are shaded. Rather than the usual single fragment invocation for each pixel covered by a primitive, multiple pixels can be shaded by a single fragment shader invocation.

Up to three methods are available to the application to change the fragment shading rate:

- [Pipeline Fragment Shading Rate](#), which allows the specification of a rate per-draw.
- [Primitive Fragment Shading Rate](#), which allows the specification of a rate per primitive, specified during shading.
- [Attachment Fragment Shading Rate](#), which allows the specification of a rate per-region of the framebuffer, specified in a specialized image attachment.

Additionally, these rates can all be specified and combined in order to adjust the overall detail in the image at each point.

This functionality can be used to focus shading efforts where higher levels of detail are needed in some parts of a scene compared to others. This can be particularly useful in high resolution rendering, or for XR contexts.

This extension also adds support for the [SPV_KHR_fragment_shading_rate](#) extension which enables setting the [primitive fragment shading rate](#), and allows querying the final shading rate from a fragment shader.

New Commands

- [vkCmdSetFragmentShadingRateKHR](#)
- [vkGetPhysicalDeviceFragmentShadingRatesKHR](#)

New Structures

- [VkPhysicalDeviceFragmentShadingRateKHR](#)
- Extending [VkGraphicsPipelineCreateInfo](#):
 - [VkPipelineFragmentShadingRateStateCreateInfoKHR](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceFragmentShadingRateFeaturesKHR](#)
- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDeviceFragmentShadingRatePropertiesKHR](#)
- Extending [VkSubpassDescription2](#):
 - [VkFragmentShadingRateAttachmentInfoKHR](#)

New Enums

- [VkFragmentShadingRateCombinerOpKHR](#)

New Enum Constants

- [VK_KHR_FRAGMENT_SHADING_RATE_EXTENSION_NAME](#)
- [VK_KHR_FRAGMENT_SHADING_RATE_SPEC_VERSION](#)
- Extending [VkAccessFlagBits](#):
 - [VK_ACCESS_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR](#)
- Extending [VkDynamicState](#):
 - [VK_DYNAMIC_STATE_FRAGMENT_SHADING_RATE_KHR](#)
- Extending [VkFormatFeatureFlagBits](#):
 - [VK_FORMAT_FEATURE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR](#)
- Extending [VkImageLayout](#):
 - [VK_IMAGE_LAYOUT_FRAGMENT_SHADING_RATE_ATTACHMENT_OPTIMAL_KHR](#)
- Extending [VkImageUsageFlagBits](#):
 - [VK_IMAGE_USAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR](#)
- Extending [VkPipelineStageFlagBits](#):
 - [VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_FRAGMENT_SHADING_RATE_ATTACHMENT_INFO_KHR](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_FEATURES_KHR](#)

- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_KHR`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADING_RATE_PROPERTIES_KHR`
- `VK_STRUCTURE_TYPE_PIPELINE_FRAGMENT_SHADING_RATE_STATE_CREATE_INFO_KHR`

If `VK_KHR_format_feature_flags2` or [Vulkan Version 1.3](#) is supported:

- Extending [VkFormatFeatureFlagBits2](#):
 - `VK_FORMAT_FEATURE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`

If [Vulkan Version 1.3](#) or `VK_KHR_dynamic_rendering` is supported:

- Extending [VkStructureType](#):
 - `VK_STRUCTURE_TYPE_RENDERING_FRAGMENT_SHADING_RATE_ATTACHMENT_INFO_KHR`

Version History

- Revision 1, 2020-05-06 (Tobias Hector)
 - Initial revision
- Revision 2, 2021-09-30 (Jon Leech)
 - Add interaction with `VK_KHR_format_feature_flags2` to `vk.xml`

VK_KHR_get_display_properties2

Name String

`VK_KHR_get_display_properties2`

Extension Type

Instance extension

Registered Extension Number

122

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_display](#)

Contact

- James Jones [\[GitHub\]](#)[cubanismo](#)

Other Extension Metadata

Last Modified Date

2017-02-21

IP Status

No known IP claims.

Contributors

- Ian Elliott, Google
- James Jones, NVIDIA

Description

This extension provides new queries for device display properties and capabilities that can be easily extended by other extensions, without introducing any further queries. This extension can be considered the [VK_KHR_display](#) equivalent of the [VK_KHR_get_physical_device_properties2](#) extension.

New Commands

- [vkGetDisplayModeProperties2KHR](#)
- [vkGetDisplayPlaneCapabilities2KHR](#)
- [vkGetPhysicalDeviceDisplayPlaneProperties2KHR](#)
- [vkGetPhysicalDeviceDisplayProperties2KHR](#)

New Structures

- [VkDisplayModeProperties2KHR](#)
- [VkDisplayPlaneCapabilities2KHR](#)
- [VkDisplayPlaneInfo2KHR](#)
- [VkDisplayPlaneProperties2KHR](#)
- [VkDisplayProperties2KHR](#)

New Enum Constants

- [VK_KHR_GET_DISPLAY_PROPERTIES_2_EXTENSION_NAME](#)
- [VK_KHR_GET_DISPLAY_PROPERTIES_2_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_DISPLAY_MODE_PROPERTIES_2_KHR](#)
 - [VK_STRUCTURE_TYPE_DISPLAY_PLANE_CAPABILITIES_2_KHR](#)
 - [VK_STRUCTURE_TYPE_DISPLAY_PLANE_INFO_2_KHR](#)
 - [VK_STRUCTURE_TYPE_DISPLAY_PLANE_PROPERTIES_2_KHR](#)
 - [VK_STRUCTURE_TYPE_DISPLAY_PROPERTIES_2_KHR](#)

Issues

1) What should this extension be named?

RESOLVED: `VK_KHR_get_display_properties2`. Other alternatives:

- `VK_KHR_display2`
- One extension, combined with `VK_KHR_surface_capabilities2`.

2) Should extensible input structs be added for these new functions:

RESOLVED:

- `vkGetPhysicalDeviceDisplayProperties2KHR`: No. The only current input is a `VkPhysicalDevice`. Other inputs would not make sense.
- `vkGetPhysicalDeviceDisplayPlaneProperties2KHR`: No. The only current input is a `VkPhysicalDevice`. Other inputs would not make sense.
- `vkGetDisplayModeProperties2KHR`: No. The only current inputs are a `VkPhysicalDevice` and a `VkDisplayModeKHR`. Other inputs would not make sense.

3) Should additional display query functions be extended?

RESOLVED:

- `vkGetDisplayPlaneSupportedDisplaysKHR`: No. Extensions should instead extend `vkGetDisplayPlaneCapabilitiesKHR()`.

Version History

- Revision 1, 2017-02-21 (James Jones)
 - Initial draft.

VK_KHR_get_surface_capabilities2

Name String

`VK_KHR_get_surface_capabilities2`

Extension Type

Instance extension

Registered Extension Number

120

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_surface](#)

Contact

- James Jones [\[GitHub\]](#)[cubanismo](#)

Other Extension Metadata

Last Modified Date

2017-02-27

IP Status

No known IP claims.

Contributors

- Ian Elliott, Google
- James Jones, NVIDIA
- Alon Or-bach, Samsung

Description

This extension provides new queries for device surface capabilities that can be easily extended by other extensions, without introducing any further queries. This extension can be considered the [VK_KHR_surface](#) equivalent of the [VK_KHR_get_physical_device_properties2](#) extension.

New Commands

- [vkGetPhysicalDeviceSurfaceCapabilities2KHR](#)
- [vkGetPhysicalDeviceSurfaceFormats2KHR](#)

New Structures

- [VkPhysicalDeviceSurfaceInfo2KHR](#)
- [VkSurfaceCapabilities2KHR](#)
- [VkSurfaceFormat2KHR](#)

New Enum Constants

- [VK_KHR_GET_SURFACE_CAPABILITIES_2_EXTENSION_NAME](#)
- [VK_KHR_GET_SURFACE_CAPABILITIES_2_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SURFACE_INFO_2_KHR](#)
 - [VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_KHR](#)
 - [VK_STRUCTURE_TYPE_SURFACE_FORMAT_2_KHR](#)

Issues

1) What should this extension be named?

RESOLVED: `VK_KHR_get_surface_capabilities2`. Other alternatives:

- `VK_KHR_surface2`
- One extension, combining a separate display-specific query extension.

2) Should additional WSI query functions be extended?

RESOLVED:

- `vkGetPhysicalDeviceSurfaceCapabilitiesKHR`: Yes. The need for this motivated the extension.
- `vkGetPhysicalDeviceSurfaceSupportKHR`: No. Currently only has boolean output. Extensions should instead extend `vkGetPhysicalDeviceSurfaceCapabilities2KHR`.
- `vkGetPhysicalDeviceSurfaceFormatsKHR`: Yes.
- `vkGetPhysicalDeviceSurfacePresentModesKHR`: No. Recent discussion concluded this introduced too much variability for applications to deal with. Extensions should instead extend `vkGetPhysicalDeviceSurfaceCapabilities2KHR`.
- `vkGetPhysicalDeviceXlibPresentationSupportKHR`: Not in this extension.
- `vkGetPhysicalDeviceXcbPresentationSupportKHR`: Not in this extension.
- `vkGetPhysicalDeviceWaylandPresentationSupportKHR`: Not in this extension.
- `vkGetPhysicalDeviceWin32PresentationSupportKHR`: Not in this extension.

Version History

- Revision 1, 2017-02-27 (James Jones)
 - Initial draft.

VK_KHR_global_priority

Name String

`VK_KHR_global_priority`

Extension Type

Device extension

Registered Extension Number

189

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_get_physical_device_properties2`

or

Vulkan Version 1.1

Deprecation State

- *Promoted* to Vulkan 1.4

Contact

- Tobias Hector [\[GitHub\]](#)[tobski](#)

Other Extension Metadata

Last Modified Date

2021-10-22

Contributors

- Tobias Hector, AMD
- Contributors to `VK_EXT_global_priority`
- Contributors to `VK_EXT_global_priority_query`

Description

In Vulkan, users can specify device-scope queue priorities. In some cases it may be useful to extend this concept to a system-wide scope. This device extension allows applications to query the global queue priorities supported by a queue family, and then set a priority when creating queues. The default queue priority is `VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_EXT`.

Implementations can report which global priority levels are treated differently by the implementation. It is intended primarily for use in system integration along with certain platform-specific priority enforcement rules.

The driver implementation will attempt to skew hardware resource allocation in favor of the higher-priority task. Therefore, higher-priority work may retain similar latency and throughput characteristics even if the system is congested with lower priority work.

The global priority level of a queue shall take precedence over the per-process queue priority (`VkDeviceQueueCreateInfo::pQueuePriorities`).

Abuse of this feature may result in starving the rest of the system from hardware resources. Therefore, the driver implementation may deny requests to acquire a priority above the default priority (`VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_EXT`) if the caller does not have sufficient privileges. In this scenario `VK_ERROR_NOT_PERMITTED_EXT` is returned.

The driver implementation may fail the queue allocation request if resources required to complete the operation have been exhausted (either by the same process or a different process). In this scenario `VK_ERROR_INITIALIZATION_FAILED` is returned.

New Structures

- Extending [VkDeviceQueueCreateInfo](#):
 - [VkDeviceQueueGlobalPriorityCreateInfoKHR](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceGlobalPriorityQueryFeaturesKHR](#)
- Extending [VkQueueFamilyProperties2](#):
 - [VkQueueFamilyGlobalPriorityPropertiesKHR](#)

New Enums

- [VkQueueGlobalPriorityKHR](#)

New Enum Constants

- [VK_KHR_GLOBAL_PRIORITY_EXTENSION_NAME](#)
- [VK_KHR_GLOBAL_PRIORITY_SPEC_VERSION](#)
- [VK_MAX_GLOBAL_PRIORITY_SIZE_KHR](#)
- Extending [VkQueueGlobalPriority](#):
 - [VK_QUEUE_GLOBAL_PRIORITY_HIGH_KHR](#)
 - [VK_QUEUE_GLOBAL_PRIORITY_LOW_KHR](#)
 - [VK_QUEUE_GLOBAL_PRIORITY_MEDIUM_KHR](#)
 - [VK_QUEUE_GLOBAL_PRIORITY_REALTIME_KHR](#)
- Extending [VkResult](#):
 - [VK_ERROR_NOT_PERMITTED_KHR](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_DEVICE_QUEUE_GLOBAL_PRIORITY_CREATE_INFO_KHR](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_GLOBAL_PRIORITY_QUERY_FEATURES_KHR](#)
 - [VK_STRUCTURE_TYPE_QUEUE_FAMILY_GLOBAL_PRIORITY_PROPERTIES_KHR](#)

Promotion to Vulkan 1.4

Functionality in this extension is included in core Vulkan 1.4 with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

Issues

1) Can we additionally query whether a caller is permitted to acquire a specific global queue priority in this extension?

RESOLVED: No. Whether a caller has enough privilege goes with the OS, and the Vulkan driver cannot really guarantee that the privilege will not change in between this query and the actual queue creation call.

2) If more than 1 queue using global priority is requested, is there a good way to know which queue is failing the device creation?

RESOLVED: No. There is not a good way at this moment, and it is also not quite actionable for the applications to know that because the information may not be accurate. Queue creation can fail because of runtime constraints like insufficient privilege or lack of resource, and the failure is not necessarily tied to that particular queue configuration requested.

Version History

- Revision 1, 2021-10-22 (Tobias Hector)
 - Initial draft

VK_KHR_incremental_present

Name String

VK_KHR_incremental_present

Extension Type

Device extension

Registered Extension Number

85

Revision

2

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_swapchain](#)

Contact

- Ian Elliott [\[GitHub\]](#)[ianelliottus](#)

Other Extension Metadata

Last Modified Date

2016-11-02

IP Status

No known IP claims.

Contributors

- Ian Elliott, Google
- Jesse Hall, Google

- Alon Or-bach, Samsung
- James Jones, NVIDIA
- Daniel Rakos, AMD
- Ray Smith, ARM
- Mika Isojarvi, Google
- Jeff Juliano, NVIDIA
- Jeff Bolz, NVIDIA

Description

This device extension extends [vkQueuePresentKHR](#), from the [VK_KHR_swapchain](#) extension, allowing an application to specify a list of rectangular, modified regions of each image to present. This should be used in situations where an application is only changing a small portion of the presentable images within a swapchain, since it enables the presentation engine to avoid wasting time presenting parts of the surface that have not changed.

This extension is leveraged from the [EGL_KHR_swap_buffers_with_damage](#) extension.

New Structures

- [VkPresentRegionKHR](#)
- [VkRectLayerKHR](#)
- Extending [VkPresentInfoKHR](#):
 - [VkPresentRegionsKHR](#)

New Enum Constants

- [VK_KHR_INCREMENTAL_PRESENT_EXTENSION_NAME](#)
- [VK_KHR_INCREMENTAL_PRESENT_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PRESENT_REGIONS_KHR](#)

Issues

1) How should we handle stereoscopic-3D swapchains? We need to add a layer for each rectangle. One approach is to create another structure containing the [VkRect2D](#) plus layer, and have [VkPresentRegionsKHR](#) point to an array of that struct. Another approach is to have two parallel arrays, [pRectangles](#) and [pLayers](#), where [pRectangles\[i\]](#) and [pLayers\[i\]](#) must be used together. Which approach should we use, and if the array of a new structure, what should that be called?

RESOLVED: Create a new structure, which is a [VkRect2D](#) plus a layer, and will be called [VkRectLayerKHR](#).

2) Where is the origin of the [VkRectLayerKHR](#)?

RESOLVED: The upper left corner of the presentable image(s) of the swapchain, per the definition of framebuffer coordinates.

3) Does the rectangular region, [VkRectLayerKHR](#), specify pixels of the swapchain's image(s), or of the surface?

RESOLVED: Of the image(s). Some presentation engines may scale the pixels of a swapchain's image(s) to the size of the surface. The size of the swapchain's image(s) will be consistent, where the size of the surface may vary over time.

4) What if all of the rectangles for a given swapchain contain a width and/or height of zero?

RESOLVED: The application is indicating that no pixels changed since the last present. The presentation engine may use such a hint and not update any pixels for the swapchain. However, all other semantics of [vkQueuePresentKHR](#) must still be honored, including waiting for semaphores to signal.

5) When the swapchain is created with [VkSwapchainCreateInfoKHR::preTransform](#) set to a value other than [VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR](#), should the rectangular region, [VkRectLayerKHR](#), be transformed to align with the [preTransform](#)?

RESOLVED: No. The rectangular region in [VkRectLayerKHR](#) should not be transformed. As such, it may not align with the extents of the swapchain's image(s). It is the responsibility of the presentation engine to transform the rectangular region. This matches the behavior of the Android presentation engine, which set the precedent.

Version History

- Revision 1, 2016-11-02 (Ian Elliott)
 - Internal revisions
- Revision 2, 2021-03-18 (Ian Elliott)
 - Clarified alignment of rectangles for presentation engines that support transformed swapchains.

VK_KHR_index_type_uint8

Name String

[VK_KHR_index_type_uint8](#)

Extension Type

Device extension

Registered Extension Number

534

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

Vulkan Version 1.1

Deprecation State

- *Promoted* to Vulkan 1.4

Contact

- Piers Daniell [[GitHub](#)]pdaniell-nv

Other Extension Metadata

Last Modified Date

2023-06-06

IP Status

No known IP claims.

Contributors

- Jeff Bolz, NVIDIA

Description

This extension allows `uint8_t` indices to be used with `vkCmdBindIndexBuffer`.

New Structures

- Extending `VkPhysicalDeviceFeatures2`, `VkDeviceCreateInfo`:
 - `VkPhysicalDeviceIndexTypeUint8FeaturesKHR`

New Enum Constants

- `VK_KHR_INDEX_TYPE_UINT8_EXTENSION_NAME`
- `VK_KHR_INDEX_TYPE_UINT8_SPEC_VERSION`
- Extending `VkIndexType`:
 - `VK_INDEX_TYPE_UINT8_KHR`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES_KHR`

Promotion to Vulkan 1.4

Functionality in this extension is included in core Vulkan 1.4 with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

Version History

- Revision 1, 2023-06-06 (Piers Daniell)
 - Internal revisions

VK_KHR_line_rasterization

Name String

VK_KHR_line_rasterization

Extension Type

Device extension

Registered Extension Number

535

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

[Vulkan Version 1.1](#)

Deprecation State

- *Promoted* to Vulkan 1.4

Contact

- Piers Daniell [\[GitHub\]pdaniell-nv](#)

Other Extension Metadata

Last Modified Date

2023-06-08

IP Status

No known IP claims.

Contributors

- Jeff Bolz, NVIDIA
- Allen Jensen, NVIDIA
- Faith Ekstrand, Intel

Description

This extension adds some line rasterization features that are commonly used in CAD applications and supported in other APIs like OpenGL. Bresenham-style line rasterization is supported, smooth rectangular lines (coverage to alpha) are supported, and stippled lines are supported for all three line rasterization modes.

New Commands

- [vkCmdSetLineStippleKHR](#)

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceLineRasterizationFeaturesKHR](#)
- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDeviceLineRasterizationPropertiesKHR](#)
- Extending [VkPipelineRasterizationStateCreateInfo](#):
 - [VkPipelineRasterizationLineStateCreateInfoKHR](#)

New Enums

- [VkLineRasterizationModeKHR](#)

New Enum Constants

- [VK_KHR_LINE_RASTERIZATION_EXTENSION_NAME](#)
- [VK_KHR_LINE_RASTERIZATION_SPEC_VERSION](#)
- Extending [VkDynamicState](#):
 - [VK_DYNAMIC_STATE_LINE_STIPPLE_KHR](#)
- Extending [VkLineRasterizationMode](#):
 - [VK_LINE_RASTERIZATION_MODE_BRESENHAM_KHR](#)
 - [VK_LINE_RASTERIZATION_MODE_DEFAULT_KHR](#)
 - [VK_LINE_RASTERIZATION_MODE_RECTANGULAR_KHR](#)
 - [VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_KHR](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES_KHR](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES_KHR](#)
 - [VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO_KHR](#)

Promotion to Vulkan 1.4

Functionality in this extension is included in core Vulkan 1.4 with the KHR suffix omitted. The

original type, enum and command names are still available as aliases of the core functionality.

When [Version 1.4](#) is supported, the [bresenhamLines](#) feature must be supported.

Issues

1) Do we need to support Bresenham-style and smooth lines with more than one rasterization sample? i.e. the equivalent of `glDisable(GL_MULTISAMPLE)` in OpenGL when the framebuffer has more than one sample?

RESOLVED: Yes. For simplicity, Bresenham line rasterization carries forward a few restrictions from OpenGL, such as not supporting per-sample shading, alpha to coverage, or alpha to one.

Version History

- Revision 1, 2019-05-09 (Jeff Bolz)
 - Initial draft

VK_KHR_object_refresh

Name String

[VK_KHR_object_refresh](#)

Extension Type

Device extension

Registered Extension Number

309

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

None

Contact

- Aidan Fabius [\[GitHub\]afabius](#)

Other Extension Metadata

Last Modified Date

2020-01-14

IP Status

No known IP claims.

Contributors

- Aidan Fabius, Core Avionics
- Mark Bellamy, ARM

Description

Many safety critical environments are required to contend with single event upsets (SEUs). These occur when a bit in a physical device's memory or register is inadvertently flipped. It is typical for host memory to include automatic error detection (EDC) or correction (ECC) on platforms where this a concern. However, device-accessible memory may not have these protections. In that case, the data must be periodically refreshed.

Unextended Vulkan provides a variety of methods to mitigate SEUs. Image and buffer objects can be bound to SEU-safe memory, and many object types can be refreshed explicitly by the application by reloading or regenerating the object's data. However, implementations may store internal object-specific data in non-SEU-safe memory, and unextended Vulkan provides no clear method to determine which object types this applies to or how to refresh that data.

This extension adds a mechanism to query which object types store implementation-internal data in device regions susceptible to SEUs, and to explicitly refresh that implementation-internal data.

New Commands

- [vkCmdRefreshObjectsKHR](#)
- [vkGetPhysicalDeviceRefreshableObjectTypesKHR](#)

New Structures

- [VkRefreshObjectKHR](#)
- [VkRefreshObjectListKHR](#)

New Enums

- [VkRefreshObjectFlagBitsKHR](#)

New Bitmasks

- [VkRefreshObjectFlagsKHR](#)

New Enum Constants

- [VK_KHR_OBJECT_REFRESH_EXTENSION_NAME](#)
- [VK_KHR_OBJECT_REFRESH_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_REFRESH_OBJECT_LIST_KHR](#)

Issues

1) Should this extension refresh object data, or validate whether or not the data has been corrupted?

RESOLVED This extension should refresh data, not validate it. This reduces application error-handling complexity, and invalid data would have to be refreshed anyway.

2) Should object refreshes be done using the host or with command buffers?

RESOLVED Object refreshes should be done with command buffers. This reduces the synchronization complexity.

3) Refresh operations will need a pipeline barrier so that subsequent commands will see the results of the refresh. What access flags and pipeline stage should apply to refresh operations? Should they use new flags and stages, or reuse an existing one?

RESOLVED Object refreshes are considered to be a transfer operation for the purposes of pipeline barriers.

4) Should this extension add a feature bit?

RESOLVED A feature bit is not necessary. In the case of this extension being promoted to core, implementations that do not support or require refreshing of any object types will return 0 for the `count` parameter of `vkGetPhysicalDeviceRefreshableObjectTypesKHR`.

Examples

None.

Version History

- Revision 1, 2020-01-14

VK_KHR_performance_query

Name String

`VK_KHR_performance_query`

Extension Type

Device extension

Registered Extension Number

117

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_get_physical_device_properties2](#)

or

[Vulkan Version 1.1](#)

API Interactions

- Interacts with [VKSC_VERSION_1_0](#)

Special Use

- [Developer tools](#)

Contact

- Alon Or-bach [\[GitHub\]](#)[alonorbach](#)

Other Extension Metadata

Last Modified Date

2019-10-08

IP Status

No known IP claims.

Contributors

- Jesse Barker, Unity Technologies
- Kenneth Benzie, Codeplay
- Jan-Harald Fredriksen, ARM
- Jeff Leger, Qualcomm
- Jesse Hall, Google
- Tobias Hector, AMD
- Neil Henning, Codeplay
- Baldur Karlsson
- Lionel Landwerlin, Intel
- Peter Lohrmann, AMD
- Alon Or-bach, Samsung
- Daniel Rakos, AMD
- Niklas Smedberg, Unity Technologies
- Igor Ostrowski, Intel

Description

The [VK_KHR_performance_query](#) extension adds a mechanism to allow querying of performance counters for use in applications and by profiling tools.

Each queue family **may** expose counters that **can** be enabled on a queue of that family. We extend

[VkQueryType](#) to add a new query type for performance queries, and chain a structure on [VkQueryPoolCreateInfo](#) to specify the performance queries to enable.

New Commands

- [vkAcquireProfilingLockKHR](#)
- [vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR](#)
- [vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR](#)
- [vkReleaseProfilingLockKHR](#)

New Structures

- [VkAcquireProfilingLockInfoKHR](#)
- [VkPerformanceCounterDescriptionKHR](#)
- [VkPerformanceCounterKHR](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDevicePerformanceQueryFeaturesKHR](#)
- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDevicePerformanceQueryPropertiesKHR](#)
- Extending [VkQueryPoolCreateInfo](#):
 - [VkQueryPoolPerformanceCreateInfoKHR](#)
- Extending [VkSubmitInfo](#), [VkSubmitInfo2](#):
 - [VkPerformanceQuerySubmitInfoKHR](#)

If Vulkan SC 1.0 is supported:

- Extending [VkDeviceCreateInfo](#):
 - [VkPerformanceQueryReservationInfoKHR](#)

New Unions

- [VkPerformanceCounterResultKHR](#)

New Enums

- [VkAcquireProfilingLockFlagBitsKHR](#)
- [VkPerformanceCounterDescriptionFlagBitsKHR](#)
- [VkPerformanceCounterScopeKHR](#)
- [VkPerformanceCounterStorageKHR](#)
- [VkPerformanceCounterUnitKHR](#)

New Bitmasks

- [VkAcquireProfilingLockFlagsKHR](#)
- [VkPerformanceCounterDescriptionFlagsKHR](#)

New Enum Constants

- [VK_KHR_PERFORMANCE_QUERY_EXTENSION_NAME](#)
- [VK_KHR_PERFORMANCE_QUERY_SPEC_VERSION](#)
- Extending [VkQueryType](#):
 - [VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_ACQUIRE_PROFILING_LOCK_INFO_KHR](#)
 - [VK_STRUCTURE_TYPE_PERFORMANCE_COUNTER_DESCRIPTION_KHR](#)
 - [VK_STRUCTURE_TYPE_PERFORMANCE_COUNTER_KHR](#)
 - [VK_STRUCTURE_TYPE_PERFORMANCE_QUERY_SUBMIT_INFO_KHR](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PERFORMANCE_QUERY_FEATURES_KHR](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PERFORMANCE_QUERY_PROPERTIES_KHR](#)
 - [VK_STRUCTURE_TYPE_QUERY_POOL_PERFORMANCE_CREATE_INFO_KHR](#)

If Vulkan SC 1.0 is supported:

- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PERFORMANCE_QUERY_RESERVATION_INFO_KHR](#)

Issues

1) Should this extension include a mechanism to begin a query in command buffer *A* and end the query in command buffer *B*?

RESOLVED No - queries are tied to command buffer creation and thus have to be encapsulated within a single command buffer.

2) Should this extension include a mechanism to begin and end queries globally on the queue, not using the existing command buffer commands?

RESOLVED No - for the same reasoning as the resolution of 1).

3) Should this extension expose counters that require multiple passes?

RESOLVED Yes - users should re-submit a command buffer with the same commands in it multiple times, specifying the pass to count as the query parameter in [VkPerformanceQuerySubmitInfoKHR](#).

4) How to handle counters across parallel workloads?

RESOLVED In the spirit of Vulkan, a counter description flag

`VK_PERFORMANCE_COUNTER_DESCRIPTION_CONCURRENTLY_IMPACTED_BIT_KHR` denotes that the accuracy of a counter result is affected by parallel workloads.

5) How to handle secondary command buffers?

RESOLVED Secondary command buffers inherit any counter pass index specified in the parent primary command buffer. Note: this is no longer an issue after change from issue 10 resolution

6) What commands does the profiling lock have to be held for?

RESOLVED For any command buffer that is being queried with a performance query pool, the profiling lock **must** be held while that command buffer is in the *recording*, *executable*, or *pending state*.

7) Should we support `vkCmdCopyQueryPoolResults`?

RESOLVED Yes.

8) Should we allow performance queries to interact with multiview?

RESOLVED Yes, but the performance queries must be performed once for each pass per view.

9) Should a `queryCount > 1` be usable for performance queries?

RESOLVED Yes. Some vendors will have costly performance counter query pool creation, and would rather if a certain set of counters were to be used multiple times that a `queryCount > 1` can be used to amortize the instantiation cost.

10) Should we introduce an indirect mechanism to set the counter pass index?

RESOLVED Specify the counter pass index at submit time instead, to avoid requiring re-recording of command buffers when multiple counter passes are needed.

Examples

The following example shows how to find what performance counters a queue family supports, setup a query pool to record these performance counters, how to add the query pool to the command buffer to record information, and how to get the results from the query pool.

```
// A previously created physical device
VkPhysicalDevice physicalDevice;

// One of the queue families our device supports
uint32_t queueFamilyIndex;

uint32_t counterCount;

// Get the count of counters supported
vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR(
    physicalDevice,
    queueFamilyIndex,
```

```

&counterCount,
NULL,
NULL);

VkPerformanceCounterKHR* counters =
    malloc(sizeof(VkPerformanceCounterKHR) * counterCount);
VkPerformanceCounterDescriptionKHR* counterDescriptions =
    malloc(sizeof(VkPerformanceCounterDescriptionKHR) * counterCount);

// Get the counters supported
vkEnumeratePhysicalDeviceQueueFamilyPerformanceQueryCountersKHR(
    physicalDevice,
    queueFamilyIndex,
    &counterCount,
    counters,
    counterDescriptions);

// Try to enable the first 8 counters
uint32_t enabledCounters[8];

const uint32_t enabledCounterCount = min(counterCount, 8));

for (uint32_t i = 0; i < enabledCounterCount; i++) {
    enabledCounters[i] = i;
}

// A previously created device that had the performanceCounterQueryPools feature
// set to VK_TRUE
VkDevice device;

VkQueryPoolPerformanceCreateInfoKHR performanceQueryCreateInfo = {
    .sType = VK_STRUCTURE_TYPE_QUERY_POOL_PERFORMANCE_CREATE_INFO_KHR,
    .pNext = NULL,

    // Specify the queue family that this performance query is performed on
    .queueFamilyIndex = queueFamilyIndex,

    // The number of counters to enable
    .counterIndexCount = enabledCounterCount,

    // The array of indices of counters to enable
    .pCounterIndices = enabledCounters
};

// Get the number of passes our counters will require.
uint32_t numPasses;

vkGetPhysicalDeviceQueueFamilyPerformanceQueryPassesKHR(
    physicalDevice,
    &performanceQueryCreateInfo,

```

```

&numPasses);

VkQueryPoolCreateInfo queryPoolCreateInfo = {
    .sType = VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO,
    .pNext = &performanceQueryCreateInfo,
    .flags = 0,
    // Using our new query type here
    .queryType = VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR,
    .queryCount = 1,
    .pipelineStatistics = 0
};

VkQueryPool queryPool;

VkResult result = vkCreateQueryPool(
    device,
    &queryPoolCreateInfo,
    NULL,
    &queryPool);

assert(VK_SUCCESS == result);

// A queue from queueFamilyIndex
VkQueue queue;

// A command buffer we want to record counters on
VkCommandBuffer commandBuffer;

VkCommandBufferBeginInfo commandBufferBeginInfo = {
    .sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO,
    .pNext = NULL,
    .flags = 0,
    .pInheritanceInfo = NULL
};

VkAcquireProfilingLockInfoKHR lockInfo = {
    .sType = VK_STRUCTURE_TYPE_ACQUIRE_PROFILING_LOCK_INFO_KHR,
    .pNext = NULL,
    .flags = 0,
    .timeout = UINT64_MAX // Wait forever for the lock
};

// Acquire the profiling lock before we record command buffers
// that will use performance queries

result = vkAcquireProfilingLockKHR(device, &lockInfo);

assert(VK_SUCCESS == result);

result = vkBeginCommandBuffer(commandBuffer, &commandBufferBeginInfo);

```

```

assert(VK_SUCCESS == result);

vkCmdResetQueryPool(
    commandBuffer,
    queryPool,
    0,
    1);

vkCmdBeginQuery(
    commandBuffer,
    queryPool,
    0,
    0);

// Perform the commands you want to get performance information on
// ...

// Perform a barrier to ensure all previous commands were complete before
// ending the query
vkCmdPipelineBarrier(commandBuffer,
    VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT,
    VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT,
    0,
    0,
    NULL,
    0,
    NULL,
    0,
    NULL);

vkCmdEndQuery(
    commandBuffer,
    queryPool,
    0);

result = vkEndCommandBuffer(commandBuffer);

assert(VK_SUCCESS == result);

for (uint32_t counterPass = 0; counterPass < numPasses; counterPass++) {

    VkPerformanceQuerySubmitInfoKHR performanceQuerySubmitInfo = {
        VK_STRUCTURE_TYPE_PERFORMANCE_QUERY_SUBMIT_INFO_KHR,
        NULL,
        counterPass
    };

    // Submit the command buffer and wait for its completion
    // ...
}

```



```

// Release the profiling lock after the command buffer is no longer in the
// pending state.
vkReleaseProfilingLockKHR(device);

result = vkResetCommandBuffer(commandBuffer, 0);

assert(VK_SUCCESS == result);

// Create an array to hold the results of all counters
VkPerformanceCounterResultKHR* recordedCounters = malloc(
    sizeof(VkPerformanceCounterResultKHR) * enabledCounterCount);

result = vkGetQueryPoolResults(
    device,
    queryPool,
    0,
    1,
    sizeof(VkPerformanceCounterResultKHR) * enabledCounterCount,
    recordedCounters,
    sizeof(VkPerformanceCounterResultKHR) * enabledCounterCount,
    NULL);

// recordedCounters is filled with our counters, we will look at one for posterity
switch (counters[0].storage) {
    case VK_PERFORMANCE_COUNTER_STORAGE_INT32:
        // use recordCounters[0].int32 to get at the counter result!
        break;
    case VK_PERFORMANCE_COUNTER_STORAGE_INT64:
        // use recordCounters[0].int64 to get at the counter result!
        break;
    case VK_PERFORMANCE_COUNTER_STORAGE_UINT32:
        // use recordCounters[0].uint32 to get at the counter result!
        break;
    case VK_PERFORMANCE_COUNTER_STORAGE_UINT64:
        // use recordCounters[0].uint64 to get at the counter result!
        break;
    case VK_PERFORMANCE_COUNTER_STORAGE_FLOAT32:
        // use recordCounters[0].float32 to get at the counter result!
        break;
    case VK_PERFORMANCE_COUNTER_STORAGE_FLOAT64:
        // use recordCounters[0].float64 to get at the counter result!
        break;
}

```

Version History

- Revision 1, 2019-10-08

VK_KHR_shader_clock

Name String

VK_KHR_shader_clock

Extension Type

Device extension

Registered Extension Number

182

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

[Vulkan Version 1.1](#)

SPIR-V Dependencies

- [SPV_KHR_shader_clock](#)

Contact

- Aaron Hagan [\[GitHub\]](#)ahagan

Other Extension Metadata

Last Modified Date

2019-4-25

IP Status

No known IP claims.

Interactions and External Dependencies

- This extension provides API support for [GL_ARB_shader_clock](#) and [GL_EXT_shader_realtime_clock](#)

Contributors

- Aaron Hagan, AMD
- Daniel Koch, NVIDIA

Description

This extension advertises the SPIR-V [ShaderClockKHR](#) capability for Vulkan, which allows a shader to query a real-time or monotonically incrementing counter at the subgroup level or across the device level. The two valid SPIR-V scopes for [OpReadClockKHR](#) are [Subgroup](#) and [Device](#).

When using GLSL source-based shading languages, the `clockRealtime*EXT()` timing functions map to the `OpReadClockKHR` instruction with a scope of `Device`, and the `clock*ARB()` timing functions map to the `OpReadClockKHR` instruction with a scope of `Subgroup`.

New Structures

- Extending `VkPhysicalDeviceFeatures2`, `VkDeviceCreateInfo`:
 - `VkPhysicalDeviceShaderClockFeaturesKHR`

New Enum Constants

- `VK_KHR_SHADER_CLOCK_EXTENSION_NAME`
- `VK_KHR_SHADER_CLOCK_SPEC_VERSION`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_CLOCK_FEATURES_KHR`

New SPIR-V Capabilities

- `ShaderClockKHR`

Version History

- Revision 1, 2019-4-25 (Aaron Hagan)
 - Initial revision

VK_KHR_shader_terminate_invocation

Name String

`VK_KHR_shader_terminate_invocation`

Extension Type

Device extension

Registered Extension Number

216

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_get_physical_device_properties2`

or

`Vulkan Version 1.1`

SPIR-V Dependencies

- [SPV_KHR_terminate_invocation](#)

Deprecation State

- *Promoted* to Vulkan 1.3

Contact

- Jesse Hall [\[GitHub\]](#)[critsec](#)

Other Extension Metadata

Last Modified Date

2020-08-11

IP Status

No known IP claims.

Contributors

- Alan Baker, Google
- Jeff Bolz, NVIDIA
- Jesse Hall, Google
- Ralph Potter, Samsung
- Tom Olson, Arm

Description

This extension adds Vulkan support for the [SPV_KHR_terminate_invocation](#) SPIR-V extension. That SPIR-V extension provides a new instruction, [OpTerminateInvocation](#), which causes a shader invocation to immediately terminate and sets the coverage of shaded samples to [0](#); only previously executed instructions will have observable effects. The [OpTerminateInvocation](#) instruction, along with the [OpDemoteToHelperInvocation](#) instruction from the [VK_EXT_shader_demote_to_helper_invocation](#) extension, together replace the [OpKill](#) instruction, which could behave like either of these instructions. [OpTerminateInvocation](#) provides the behavior required by the GLSL [discard](#) statement, and should be used when available by GLSL compilers and applications that need the GLSL [discard](#) behavior.

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceShaderTerminateInvocationFeaturesKHR](#)

New Enum Constants

- [VK_KHR_SHADER_TERMINATE_INVOCATION_EXTENSION_NAME](#)
- [VK_KHR_SHADER_TERMINATE_INVOCATION_SPEC_VERSION](#)
- Extending [VkStructureType](#):

- [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_TERMINATE_INVOCATION_FEATURES_KHR](#)

Promotion to Vulkan 1.3

Vulkan APIs in this extension are included in core Vulkan 1.3, with the KHR suffix omitted. External interactions defined by this extension, such as SPIR-V token names, retain their original names. The original Vulkan API names are still available as aliases of the core functionality.

Version History

- Revision 1, 2020-08-11 (Jesse Hall)

VK_KHR_shared_presentable_image

Name String

[VK_KHR_shared_presentable_image](#)

Extension Type

Device extension

Registered Extension Number

112

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_swapchain](#)

and

[VK_KHR_get_surface_capabilities2](#)

and

[VK_KHR_get_physical_device_properties2](#)

or

[Vulkan Version 1.1](#)

Contact

- Alon Or-bach [\[GitHub\]alonorbach](#)

Other Extension Metadata

Last Modified Date

2017-03-20

IP Status

No known IP claims.

Contributors

- Alon Or-bach, Samsung Electronics
- Ian Elliott, Google
- Jesse Hall, Google
- Pablo Ceballos, Google
- Chris Forbes, Google
- Jeff Juliano, NVIDIA
- James Jones, NVIDIA
- Daniel Rakos, AMD
- Tobias Hector, Imagination Technologies
- Graham Connor, Imagination Technologies
- Michael Worcester, Imagination Technologies
- Cass Everitt, Oculus
- Johannes Van Waveren, Oculus

Description

This extension extends [VK_KHR_swapchain](#) to enable creation of a shared presentable image. This allows the application to use the image while the presentation engine is accessing it, in order to reduce the latency between rendering and presentation.

New Commands

- [vkGetSwapchainStatusKHR](#)

New Structures

- Extending [VkSurfaceCapabilities2KHR](#):
 - [VkSharedPresentSurfaceCapabilitiesKHR](#)

New Enum Constants

- [VK_KHR_SHARED_PRESENTABLE_IMAGE_EXTENSION_NAME](#)
- [VK_KHR_SHARED_PRESENTABLE_IMAGE_SPEC_VERSION](#)
- Extending [VkImageLayout](#):
 - [VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR](#)
- Extending [VkPresentModeKHR](#):
 - [VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR](#)
 - [VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_SHARED_PRESENT_SURFACE_CAPABILITIES_KHR](#)

Issues

1) Should we allow a Vulkan WSI swapchain to toggle between normal usage and shared presentation usage?

RESOLVED: No. WSI swapchains are typically recreated with new properties instead of having their properties changed. This can also save resources, assuming that fewer images are needed for shared presentation, and assuming that most VR applications do not need to switch between normal and shared usage.

2) Should we have a query for determining how the presentation engine refresh is triggered?

RESOLVED: Yes. This is done via which presentation modes a surface supports.

3) Should the object representing a shared presentable image be an extension of a [VkSwapchainKHR](#) or a separate object?

RESOLVED: Extension of a swapchain due to overlap in creation properties and to allow common functionality between shared and normal presentable images and swapchains.

4) What should we call the extension and the new structures it creates?

RESOLVED: Shared presentable image / shared present.

5) Should the `minImageCount` and `presentMode` values of the [VkSwapchainCreateInfoKHR](#) be ignored, or required to be compatible values?

RESOLVED: `minImageCount` must be 1, and `presentMode` should be set to either `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` or `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`.

6) What should the layout of the shared presentable image be?

RESOLVED: After acquiring the shared presentable image, the application must transition it to the `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR` layout prior to it being used. After this initial transition, any image usage that was requested during swapchain creation **can** be performed on the image without layout transitions being performed.

7) Do we need a new API for the trigger to refresh new content?

RESOLVED: [vkQueuePresentKHR](#) to act as API to trigger a refresh, as will allow combination with other compatible extensions to [vkQueuePresentKHR](#).

8) How should an application detect a `VK_ERROR_OUT_OF_DATE_KHR` error on a swapchain using the `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR` present mode?

RESOLVED: Introduce [vkGetSwapchainStatusKHR](#) to allow applications to query the status of a swapchain using a shared presentation mode.

9) What should subsequent calls to [vkQueuePresentKHR](#) for `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR` swapchains be defined to do?

RESOLVED: State that implementations may use it as a hint for updated content.

10) Can the ownership of a shared presentable image be transferred to a different queue?

RESOLVED: No. It is not possible to transfer ownership of a shared presentable image obtained from a swapchain created using `VK_SHARING_MODE_EXCLUSIVE` after it has been presented.

11) How should `vkQueueSubmit` behave if a command buffer uses an image from a `VK_ERROR_OUT_OF_DATE_KHR` swapchain?

RESOLVED: `vkQueueSubmit` is expected to return the `VK_ERROR_DEVICE_LOST` error.

12) Can Vulkan provide any guarantee on the order of rendering, to enable beam chasing?

RESOLVED: This could be achieved via use of render passes to ensure strip rendering.

Version History

- Revision 1, 2017-03-20 (Alon Or-bach)
 - Internal revisions

VK_KHR_surface

Name String

`VK_KHR_surface`

Extension Type

Instance extension

Registered Extension Number

1

Revision

25

Ratification Status

Ratified

Extension and Version Dependencies

None

Contact

- James Jones [\[GitHub\]cubanismo](#)
- Ian Elliott [\[GitHub\]ianelliottus](#)

Other Extension Metadata

Last Modified Date

2016-08-25

IP Status

No known IP claims.

Contributors

- Patrick Doane, Blizzard
- Ian Elliott, LunarG
- Jesse Hall, Google
- James Jones, NVIDIA
- David Mao, AMD
- Norbert Nopper, Freescale
- Alon Or-bach, Samsung
- Daniel Rakos, AMD
- Graham Sellers, AMD
- Jeff Vigil, Qualcomm
- Chia-I Wu, LunarG
- Faith Ekstrand, Intel

Description

The `VK_KHR_surface` extension is an instance extension. It introduces `VkSurfaceKHR` objects, which abstract native platform surface or window objects for use with Vulkan. It also provides a way to determine whether a queue family in a physical device supports presenting to particular surface.

Separate extensions for each platform provide the mechanisms for creating `VkSurfaceKHR` objects, but once created they may be used in this and other platform-independent extensions, in particular the `VK_KHR_swapchain` extension.

New Object Types

- `VkSurfaceKHR`

New Commands

- `vkDestroySurfaceKHR`
- `vkGetPhysicalDeviceSurfaceCapabilitiesKHR`
- `vkGetPhysicalDeviceSurfaceFormatsKHR`
- `vkGetPhysicalDeviceSurfacePresentModesKHR`
- `vkGetPhysicalDeviceSurfaceSupportKHR`

New Structures

- `VkSurfaceCapabilitiesKHR`
- `VkSurfaceFormatKHR`

New Enums

- [VkColorSpaceKHR](#)
- [VkCompositeAlphaFlagBitsKHR](#)
- [VkPresentModeKHR](#)
- [VkSurfaceTransformFlagBitsKHR](#)

New Bitmasks

- [VkCompositeAlphaFlagsKHR](#)

New Enum Constants

- [VK_KHR_SURFACE_EXTENSION_NAME](#)
- [VK_KHR_SURFACE_SPEC_VERSION](#)
- Extending [VkObjectType](#):
 - [VK_OBJECT_TYPE_SURFACE_KHR](#)
- Extending [VkResult](#):
 - [VK_ERROR_NATIVE_WINDOW_IN_USE_KHR](#)
 - [VK_ERROR_SURFACE_LOST_KHR](#)

Issues

1) Should this extension include a method to query whether a physical device supports presenting to a specific window or native surface on a given platform?

RESOLVED: Yes. Without this, applications would need to create a device instance to determine whether a particular window can be presented to. Knowing that a device supports presentation to a platform in general is not sufficient, as a single machine might support multiple seats, or instances of the platform that each use different underlying physical devices. Additionally, on some platforms, such as the X Window System, different drivers and devices might be used for different windows depending on which section of the desktop they exist on.

2) Should the [vkGetPhysicalDeviceSurfaceCapabilitiesKHR](#), [vkGetPhysicalDeviceSurfaceFormatsKHR](#), and [vkGetPhysicalDeviceSurfacePresentModesKHR](#) functions be in this extension and operate on physical devices, rather than being in [VK_KHR_swapchain](#) (i.e. device extension) and being dependent on [VkDevice](#)?

RESOLVED: Yes. While it might be useful to depend on [VkDevice](#) (and therefore on enabled extensions and features) for the queries, Vulkan was released only with the [VkPhysicalDevice](#) versions. Many cases can be resolved by a Valid Usage statement, and/or by a separate [pNext](#) chain version of the query structure specific to a given extension or parameters, via extensible versions of the queries: [vkGetPhysicalDeviceSurfaceCapabilities2KHR](#), and [vkGetPhysicalDeviceSurfaceFormats2KHR](#).

3) Should Vulkan support Xlib or XCB as the API for accessing the X Window System platform?

RESOLVED: Both. XCB is a more modern and efficient API, but Xlib usage is deeply ingrained in many applications and likely will remain in use for the foreseeable future. Not all drivers necessarily need to support both, but including both as options in the core specification will probably encourage support, which should in turn ease adoption of the Vulkan API in older codebases. Additionally, the performance improvements possible with XCB likely will not have a measurable impact on the performance of Vulkan presentation and other minimal window system interactions defined here.

4) Should the GBM platform be included in the list of platform enums?

RESOLVED: Deferred, and will be addressed with a platform-specific extension to be written in the future.

Version History

- Revision 1, 2015-05-20 (James Jones)
 - Initial draft, based on LunarG KHR spec, other KHR specs, patches attached to bugs.
- Revision 2, 2015-05-22 (Ian Elliott)
 - Created initial Description section.
 - Removed query for whether a platform requires the use of a queue for presentation, since it was decided that presentation will always be modeled as being part of the queue.
 - Fixed typos and other minor mistakes.
- Revision 3, 2015-05-26 (Ian Elliott)
 - Improved the Description section.
- Revision 4, 2015-05-27 (James Jones)
 - Fixed compilation errors in example code.
- Revision 5, 2015-06-01 (James Jones)
 - Added issues 1 and 2 and made related spec updates.
- Revision 6, 2015-06-01 (James Jones)
 - Merged the platform type mappings table previously removed from VK_KHR_swapchain with the platform description table in this spec.
 - Added issues 3 and 4 documenting choices made when building the initial list of native platforms supported.
- Revision 7, 2015-06-11 (Ian Elliott)
 - Updated table 1 per input from the KHR TSG.
 - Updated issue 4 (GBM) per discussion with Daniel Stone. He will create a platform-specific extension sometime in the future.
- Revision 8, 2015-06-17 (James Jones)
 - Updated enum-extending values using new convention.
 - Fixed the value of VK_SURFACE_PLATFORM_INFO_TYPE_SUPPORTED_KHR.

- Revision 9, 2015-06-17 (James Jones)
 - Rebased on Vulkan API version 126.
- Revision 10, 2015-06-18 (James Jones)
 - Marked issues 2 and 3 resolved.
- Revision 11, 2015-06-23 (Ian Elliott)
 - Examples now show use of function pointers for extension functions.
 - Eliminated extraneous whitespace.
- Revision 12, 2015-07-07 (Daniel Rakos)
 - Added error section describing when each error is expected to be reported.
 - Replaced the term “queue node index” with “queue family index” in the spec as that is the agreed term to be used in the latest version of the core header and spec.
 - Replaced `bool32_t` with `VkBool32`.
- Revision 13, 2015-08-06 (Daniel Rakos)
 - Updated spec against latest core API header version.
- Revision 14, 2015-08-20 (Ian Elliott)
 - Renamed this extension and all of its enumerations, types, functions, etc. This makes it compliant with the proposed standard for Vulkan extensions.
 - Switched from “revision” to “version”, including use of the `VK_MAKE_VERSION` macro in the header file.
 - Did miscellaneous cleanup, etc.
- Revision 15, 2015-08-20 (Ian Elliott—porting a 2015-07-29 change from James Jones)
 - Moved the surface transform enums here from `VK_WSI_swapchain` so they could be reused by `VK_WSI_display`.
- Revision 16, 2015-09-01 (James Jones)
 - Restore single-field revision number.
- Revision 17, 2015-09-01 (James Jones)
 - Fix example code compilation errors.
- Revision 18, 2015-09-26 (Jesse Hall)
 - Replaced `VkSurfaceDescriptionKHR` with the `VkSurfaceKHR` object, which is created via layered extensions. Added `VkDestroySurfaceKHR`.
- Revision 19, 2015-09-28 (Jesse Hall)
 - Renamed from `VK_EXT_KHR_swapchain` to `VK_EXT_KHR_surface`.
- Revision 20, 2015-09-30 (Jeff Vigil)
 - Add error result `VK_ERROR_SURFACE_LOST_KHR`.
- Revision 21, 2015-10-15 (Daniel Rakos)
 - Updated the resolution of issue #2 and include the surface capability queries in this

extension.

- Renamed SurfaceProperties to SurfaceCapabilities as it better reflects that the values returned are the capabilities of the surface on a particular device.
- Other minor cleanup and consistency changes.
- Revision 22, 2015-10-26 (Ian Elliott)
 - Renamed from VK_EXT_KHR_surface to VK_KHR_surface.
- Revision 23, 2015-11-03 (Daniel Rakos)
 - Added allocation callbacks to vkDestroySurfaceKHR.
- Revision 24, 2015-11-10 (Jesse Hall)
 - Removed VkSurfaceTransformKHR. Use VkSurfaceTransformFlagBitsKHR instead.
 - Rename VkSurfaceCapabilitiesKHR member maxImageArraySize to maxImageArrayLayers.
- Revision 25, 2016-01-14 (James Jones)
 - Moved VK_ERROR_NATIVE_WINDOW_IN_USE_KHR from the VK_KHR_android_surface to the VK_KHR_surface extension.
- 2016-08-23 (Ian Elliott)
 - Update the example code, to not have so many characters per line, and to split out a new example to show how to obtain function pointers.
- 2016-08-25 (Ian Elliott)
 - A note was added at the beginning of the example code, stating that it will be removed from future versions of the appendix.

VK_KHR_swapchain

Name String

VK_KHR_swapchain

Extension Type

Device extension

Registered Extension Number

2

Revision

70

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_surface

API Interactions

- Interacts with [VK_VERSION_1_1](#)

Contact

- James Jones [\[GitHub\]](#)[cubanismo](#)
- Ian Elliott [\[GitHub\]](#)[ianelliottus](#)

Other Extension Metadata

Last Modified Date

2017-10-06

IP Status

No known IP claims.

Interactions and External Dependencies

- Interacts with Vulkan 1.1

Contributors

- Patrick Doane, Blizzard
- Ian Elliott, LunarG
- Jesse Hall, Google
- Mathias Heyer, NVIDIA
- James Jones, NVIDIA
- David Mao, AMD
- Norbert Nopper, Freescale
- Alon Or-bach, Samsung
- Daniel Rakos, AMD
- Graham Sellers, AMD
- Jeff Vigil, Qualcomm
- Chia-I Wu, LunarG
- Faith Ekstrand, Intel
- Matthaeus G. Chajdas, AMD
- Ray Smith, ARM

Description

The [VK_KHR_swapchain](#) extension is the device-level companion to the [VK_KHR_surface](#) extension. It introduces [VkSwapchainKHR](#) objects, which provide the ability to present rendering results to a surface.

New Object Types

- [VkSwapchainKHR](#)

New Commands

- [vkAcquireNextImageKHR](#)
- [vkCreateSwapchainKHR](#)
- [vkDestroySwapchainKHR](#)
- [vkGetSwapchainImagesKHR](#)
- [vkQueuePresentKHR](#)

If [Vulkan Version 1.1](#) is supported:

- [vkAcquireNextImage2KHR](#)
- [vkGetDeviceGroupPresentCapabilitiesKHR](#)
- [vkGetDeviceGroupSurfacePresentModesKHR](#)
- [vkGetPhysicalDevicePresentRectanglesKHR](#)

New Structures

- [VkPresentInfoKHR](#)
- [VkSwapchainCreateInfoKHR](#)

If [Vulkan Version 1.1](#) is supported:

- [VkAcquireNextImageInfoKHR](#)
- [VkDeviceGroupPresentCapabilitiesKHR](#)
- Extending [VkBindImageMemoryInfo](#):
 - [VkBindImageMemorySwapchainInfoKHR](#)
- Extending [VkImageCreateInfo](#):
 - [VkImageSwapchainCreateInfoKHR](#)
- Extending [VkPresentInfoKHR](#):
 - [VkDeviceGroupPresentInfoKHR](#)
- Extending [VkSwapchainCreateInfoKHR](#):
 - [VkDeviceGroupSwapchainCreateInfoKHR](#)

New Enums

- [VkSwapchainCreateFlagBitsKHR](#)

If [Vulkan Version 1.1](#) is supported:

- [VkDeviceGroupPresentModeFlagBitsKHR](#)

New Bitmasks

- [VkSwapchainCreateFlagsKHR](#)

If [Vulkan Version 1.1](#) is supported:

- [VkDeviceGroupPresentModeFlagsKHR](#)

New Enum Constants

- [VK_KHR_SWAPCHAIN_EXTENSION_NAME](#)
- [VK_KHR_SWAPCHAIN_SPEC_VERSION](#)
- Extending [VkImageLayout](#):
 - [VK_IMAGE_LAYOUT_PRESENT_SRC_KHR](#)
- Extending [VkObjectType](#):
 - [VK_OBJECT_TYPE_SWAPCHAIN_KHR](#)
- Extending [VkResult](#):
 - [VK_ERROR_OUT_OF_DATE_KHR](#)
 - [VK_SUBOPTIMAL_KHR](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PRESENT_INFO_KHR](#)
 - [VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR](#)

If [Vulkan Version 1.1](#) is supported:

- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_ACQUIRE_NEXT_IMAGE_INFO_KHR](#)
 - [VK_STRUCTURE_TYPE_BIND_IMAGE_MEMORY_SWAPCHAIN_INFO_KHR](#)
 - [VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_CAPABILITIES_KHR](#)
 - [VK_STRUCTURE_TYPE_DEVICE_GROUP_PRESENT_INFO_KHR](#)
 - [VK_STRUCTURE_TYPE_DEVICE_GROUP_SWAPCHAIN_CREATE_INFO_KHR](#)
 - [VK_STRUCTURE_TYPE_IMAGE_SWAPCHAIN_CREATE_INFO_KHR](#)
- Extending [VkSwapchainCreateFlagBitsKHR](#):
 - [VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR](#)
 - [VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR](#)

Issues

1) Does this extension allow the application to specify the memory backing of the presentable images?

RESOLVED: No. Unlike standard images, the implementation will allocate the memory backing of

the presentable image.

2) What operations are allowed on presentable images?

RESOLVED: This is determined by the image usage flags specified when creating the presentable image's swapchain.

3) Does this extension support MSAA presentable images?

RESOLVED: No. Presentable images are always single-sampled. Multi-sampled rendering must use regular images. To present the rendering results the application must manually resolve the multi-sampled image to a single-sampled presentable image prior to presentation.

4) Does this extension support stereo/multi-view presentable images?

RESOLVED: Yes. The number of views associated with a presentable image is determined by the `imageArrayLayers` specified when creating a swapchain. All presentable images in a given swapchain use the same array size.

5) Are the layers of stereo presentable images half-sized?

RESOLVED: No. The image extents always match those requested by the application.

6) Do the “present” and “acquire next image” commands operate on a queue? If not, do they need to include explicit semaphore objects to interlock them with queue operations?

RESOLVED: The present command operates on a queue. The image ownership operation it represents happens in order with other operations on the queue, so no explicit semaphore object is required to synchronize its actions.

Applications may want to acquire the next image in separate threads from those in which they manage their queue, or in multiple threads. To make such usage easier, the acquire next image command takes a semaphore to signal as a method of explicit synchronization. The application must later queue a wait for this semaphore before queuing execution of any commands using the image.

7) Does `vkAcquireNextImageKHR` block if no images are available?

RESOLVED: The command takes a timeout parameter. Special values for the timeout are 0, which makes the call a non-blocking operation, and `UINT64_MAX`, which blocks indefinitely. Values in between will block for up to the specified time. The call will return when an image becomes available or an error occurs. It may, but is not required to, return before the specified timeout expires if the swapchain becomes out of date.

8) Can multiple presents be queued using one `vkQueuePresentKHR` call?

RESOLVED: Yes. `VkPresentInfoKHR` contains a list of swapchains and corresponding image indices that will be presented. When supported, all presentations queued with a single `vkQueuePresentKHR` call will be applied atomically as one operation. The same swapchain must not appear in the list more than once. Later extensions may provide applications stronger guarantees of atomicity for such present operations, and/or allow them to query whether atomic

presentation of a particular group of swapchains is possible.

9) How do the presentation and acquire next image functions notify the application the targeted surface has changed?

RESOLVED: Two new result codes are introduced for this purpose:

- **VK_SUBOPTIMAL_KHR** - Presentation will still succeed, subject to the window resize behavior, but the swapchain is no longer configured optimally for the surface it targets. Applications should query updated surface information and recreate their swapchain at the next convenient opportunity.
- **VK_ERROR_OUT_OF_DATE_KHR** - Failure. The swapchain is no longer compatible with the surface it targets. The application must query updated surface information and recreate the swapchain before presentation will succeed.

These can be returned by both [vkAcquireNextImageKHR](#) and [vkQueuePresentKHR](#).

10) Does the [vkAcquireNextImageKHR](#) command return a semaphore to the application via an output parameter, or accept a semaphore to signal from the application as an object handle parameter?

RESOLVED: Accept a semaphore to signal as an object handle. This avoids the need to specify whether the application must destroy the semaphore or whether it is owned by the swapchain, and if the latter, what its lifetime is and whether it can be reused for other operations once it is received from [vkAcquireNextImageKHR](#).

11) What types of swapchain queuing behavior should be exposed? Options include swap interval specification, mailbox/most recent vs. FIFO queue management, targeting specific vertical blank intervals or absolute times for a given present operation, and probably others. For some of these, whether they are specified at swapchain creation time or as per-present parameters needs to be decided as well.

RESOLVED: The base swapchain extension will expose 3 possible behaviors (of which, FIFO will always be supported):

- Immediate present: Does not wait for vertical blanking period to update the current image, likely resulting in visible tearing. No internal queue is used. Present requests are applied immediately.
- Mailbox queue: Waits for the next vertical blanking period to update the current image. No tearing should be observed. An internal single-entry queue is used to hold pending presentation requests. If the queue is full when a new presentation request is received, the new request replaces the existing entry, and any images associated with the prior entry become available for reuse by the application.
- FIFO queue: Waits for the next vertical blanking period to update the current image. No tearing should be observed. An internal queue containing **numSwapchainImages** - 1 entries is used to hold pending presentation requests. New requests are appended to the end of the queue, and one request is removed from the beginning of the queue and processed during each vertical blanking period in which the queue is non-empty

Not all surfaces will support all of these modes, so the modes supported will be returned using a surface information query. All surfaces must support the FIFO queue mode. Applications must choose one of these modes up front when creating a swapchain. Switching modes can be accomplished by recreating the swapchain.

12) Can `VK_PRESENT_MODE_MAILBOX_KHR` provide non-blocking guarantees for `vkAcquireNextImageKHR`? If so, what is the proper criteria?

RESOLVED: Yes. The difficulty is not immediately obvious here. Naively, if at least 3 images are requested, mailbox mode should always have an image available for the application if the application does not own any images when the call to `vkAcquireNextImageKHR` was made. However, some presentation engines may have more than one “current” image, and would still need to block in some cases. The right requirement appears to be that if the application allocates the surface’s minimum number of images + 1 then it is guaranteed non-blocking behavior when it does not currently own any images.

13) Is there a way to create and initialize a new swapchain for a surface that has generated a `VK_SUBOPTIMAL_KHR` return code while still using the old swapchain?

RESOLVED: Not as part of this specification. This could be useful to allow the application to create an “optimal” replacement swapchain and rebuild all its command buffers using it in a background thread at a low priority while continuing to use the “suboptimal” swapchain in the main thread. It could probably use the same “atomic replace” semantics proposed for recreating direct-to-device swapchains without incurring a mode switch. However, after discussion, it was determined some platforms probably could not support concurrent swapchains for the same surface though, so this will be left out of the base KHR extensions. A future extension could add this for platforms where it is supported.

14) Should there be a special value for `VkSurfaceCapabilitiesKHR::maxImageCount` to indicate there are no practical limits on the number of images in a swapchain?

RESOLVED: Yes. There will often be cases where there is no practical limit to the number of images in a swapchain other than the amount of available resources (i.e., memory) in the system. Trying to derive a hard limit from things like memory size is prone to failure. It is better in such cases to leave it to applications to figure such soft limits out via trial/failure iterations.

15) Should there be a special value for `VkSurfaceCapabilitiesKHR::currentExtent` to indicate the size of the platform surface is undefined?

RESOLVED: Yes. On some platforms (Wayland, for example), the surface size is defined by the images presented to it rather than the other way around.

16) Should there be a special value for `VkSurfaceCapabilitiesKHR::maxImageExtent` to indicate there is no practical limit on the surface size?

RESOLVED: No. It seems unlikely such a system would exist. 0 could be used to indicate the platform places no limits on the extents beyond those imposed by Vulkan for normal images, but this query could just as easily return those same limits, so a special “unlimited” value does not seem useful for this field.

17) How should surface rotation and mirroring be exposed to applications? How do they specify rotation and mirroring transforms applied prior to presentation?

RESOLVED: Applications can query both the supported and current transforms of a surface. Both are specified relative to the device's "natural" display rotation and direction. The supported transforms indicate which orientations the presentation engine accepts images in. For example, a presentation engine that does not support transforming surfaces as part of presentation, and which is presenting to a surface that is displayed with a 90-degree rotation, would return only one supported transform bit: `VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR`. Applications must transform their rendering by the transform they specify when creating the swapchain in `preTransform` field.

18) Can surfaces ever not support `VK_MIRROR_NONE`? Can they support vertical and horizontal mirroring simultaneously? Relatedly, should `VK_MIRROR_NONE_BIT` be zero, or bit one, and should applications be allowed to specify multiple pre and current mirror transform bits, or exactly one?

RESOLVED: Since some platforms may not support presenting with a transform other than the native window's current transform, and prerotation/mirroring are specified relative to the device's natural rotation and direction, rather than relative to the surface's current rotation and direction, it is necessary to express lack of support for no mirroring. To allow this, the `MIRROR_NONE` enum must occupy a bit in the flags. Since `MIRROR_NONE` must be a bit in the bitmask rather than a bitmask with no values set, allowing more than one bit to be set in the bitmask would make it possible to describe undefined transforms such as `VK_MIRROR_NONE_BIT | VK_MIRROR_HORIZONTAL_BIT`, or a transform that includes both "no mirroring" and "horizontal mirroring" simultaneously. Therefore, it is desirable to allow specifying all supported mirroring transforms using only one bit. The question then becomes, should there be a `VK_MIRROR_HORIZONTAL_AND_VERTICAL_BIT` to represent a simultaneous horizontal and vertical mirror transform? However, such a transform is equivalent to a 180 degree rotation, so presentation engines and applications that wish to support or use such a transform can express it through rotation instead. Therefore, 3 exclusive bits are sufficient to express all needed mirroring transforms.

19) Should support for sRGB be required?

RESOLVED: In the advent of UHD and HDR display devices, proper color space information is vital to the display pipeline represented by the swapchain. The application can discover the supported format/color-space pairs and select a pair most suited to its rendering needs. Currently only the sRGB color space is supported, future extensions may provide support for more color spaces. See issues 23 and 24.

20) Is there a mechanism to modify or replace an existing swapchain with one targeting the same surface?

RESOLVED: Yes. This is described above in the text.

21) Should there be a way to set prerotation and mirroring using native APIs when presenting using a Vulkan swapchain?

RESOLVED: Yes. The transforms that can be expressed in this extension are a subset of those possible on native platforms. If a platform exposes a method to specify the transform of presented images for a given surface using native methods and exposes more transforms or other properties for surfaces than Vulkan supports, it might be impossible, difficult, or inconvenient to set some of

those properties using Vulkan KHR extensions and some using the native interfaces. To avoid overwriting properties set using native commands when presenting using a Vulkan swapchain, the application can set the pretransform to “inherit”, in which case the current native properties will be used, or if none are available, a platform-specific default will be used. Platforms that do not specify a reasonable default or do not provide native mechanisms to specify such transforms should not include the inherit bits in the `supportedTransforms` bitmask they return in `VkSurfaceCapabilitiesKHR`.

22) Should the content of presentable images be clipped by objects obscuring their target surface?

RESOLVED: Applications can choose which behavior they prefer. Allowing the content to be clipped could enable more efficient presentation methods on some platforms, but some applications might rely on the content of presentable images to perform techniques such as partial updates or motion blurs.

23) What is the purpose of specifying a `VkColorSpaceKHR` along with `VkFormat` when creating a swapchain?

RESOLVED: While Vulkan itself is color space agnostic (e.g. even the meaning of R, G, B and A can be freely defined by the rendering application), the swapchain eventually will have to present the images on a display device with specific color reproduction characteristics. If any color space transformations are necessary before an image can be displayed, the color space of the presented image must be known to the swapchain. A swapchain will only support a restricted set of color format and -space pairs. This set can be discovered via `vkGetPhysicalDeviceSurfaceFormatsKHR`. As it can be expected that most display devices support the sRGB color space, at least one format/color-space pair has to be exposed, where the color space is `VK_COLOR_SPACE_SRGB_NONLINEAR_KHR`.

24) How are sRGB formats and the sRGB color space related?

RESOLVED: While Vulkan exposes a number of SRGB texture formats, using such formats does not guarantee working in a specific color space. It merely means that the hardware can directly support applying the non-linear transfer functions defined by the sRGB standard color space when reading from or writing to images of those formats. Still, it is unlikely that a swapchain will expose a `*_SRGB` format along with any color space other than `VK_COLOR_SPACE_SRGB_NONLINEAR_KHR`.

On the other hand, non-`*_SRGB` formats will be very likely exposed in pair with a SRGB color space. This means, the hardware will not apply any transfer function when reading from or writing to such images, yet they will still be presented on a device with sRGB display characteristics. In this case the application is responsible for applying the transfer function, for instance by using shader math.

25) How are the lifetimes of surfaces and swapchains targeting them related?

RESOLVED: A surface must outlive any swapchains targeting it. A `VkSurfaceKHR` owns the binding of the native window to the Vulkan driver.

26) How can the application control the way the alpha component of swapchain images is treated by the presentation engine during compositing?

RESOLVED: We should add new enum values to allow the application to negotiate with the

presentation engine on how to treat image alpha values during the compositing process. Since not all platforms can practically control this through the Vulkan driver, a value of `VK_COMPOSITE_ALPHA_INHERIT_BIT_KHR` is provided like for surface transforms.

27) Is `vkCreateSwapchainKHR` the right function to return `VK_ERROR_NATIVE_WINDOW_IN_USE_KHR`, or should the various platform-specific `VkSurfaceKHR` factory functions catch this error earlier?

RESOLVED: For most platforms, the `VkSurfaceKHR` structure is a simple container holding the data that identifies a native window or other object representing a surface on a particular platform. For the surface factory functions to return this error, they would likely need to register a reference on the native objects with the native display server somehow, and ensure no other such references exist. Surfaces were not intended to be that heavyweight.

Swapchains are intended to be the objects that directly manipulate native windows and communicate with the native presentation mechanisms. Swapchains will already need to communicate with the native display server to negotiate allocation and/or presentation of presentable images for a native surface. Therefore, it makes more sense for swapchain creation to be the point at which native object exclusivity is enforced. Platforms may choose to enforce further restrictions on the number of `VkSurfaceKHR` objects that may be created for the same native window if such a requirement makes sense on a particular platform, but a global requirement is only sensible at the swapchain level.

Version History

- Revision 1, 2015-05-20 (James Jones)
 - Initial draft, based on LunarG KHR spec, other KHR specs, patches attached to bugs.
- Revision 2, 2015-05-22 (Ian Elliott)
 - Made many agreed-upon changes from 2015-05-21 KHR TSG meeting. This includes using only a queue for presentation, and having an explicit function to acquire the next image.
 - Fixed typos and other minor mistakes.
- Revision 3, 2015-05-26 (Ian Elliott)
 - Improved the Description section.
 - Added or resolved issues that were found in improving the Description. For example, `pSurfaceDescription` is used consistently, instead of sometimes using `pSurface`.
- Revision 4, 2015-05-27 (James Jones)
 - Fixed some grammatical errors and typos
 - Filled in the description of `imageUseFlags` when creating a swapchain.
 - Added a description of `swapInterval`.
 - Replaced the paragraph describing the order of operations on a queue for image ownership and presentation.
- Revision 5, 2015-05-27 (James Jones)
 - Imported relevant issues from the (abandoned) `vk_wsi_persistent_swapchain_images` extension.

- Added issues 6 and 7, regarding behavior of the acquire next image and present commands with respect to queues.
- Updated spec language and examples to align with proposed resolutions to issues 6 and 7.
- Revision 6, 2015-05-27 (James Jones)
 - Added issue 8, regarding atomic presentation of multiple swapchains
 - Updated spec language and examples to align with proposed resolution to issue 8.
- Revision 7, 2015-05-27 (James Jones)
 - Fixed compilation errors in example code, and made related spec fixes.
- Revision 8, 2015-05-27 (James Jones)
 - Added issue 9, and the related VK_SUBOPTIMAL_KHR result code.
 - Renamed VK_OUT_OF_DATE_KHR to VK_ERROR_OUT_OF_DATE_KHR.
- Revision 9, 2015-05-27 (James Jones)
 - Added inline proposed resolutions (marked with [JRJ]) to some XXX questions/issues. These should be moved to the issues section in a subsequent update if the proposals are adopted.
- Revision 10, 2015-05-28 (James Jones)
 - Converted vkAcquireNextImageKHR back to a non-queue operation that uses a VkSemaphore object for explicit synchronization.
 - Added issue 10 to determine whether vkAcquireNextImageKHR generates or returns semaphores, or whether it operates on a semaphore provided by the application.
- Revision 11, 2015-05-28 (James Jones)
 - Marked issues 6, 7, and 8 resolved.
 - Renamed VkSurfaceCapabilityPropertiesKHR to VkSurfacePropertiesKHR to better convey the mutable nature of the information it contains.
- Revision 12, 2015-05-28 (James Jones)
 - Added issue 11 with a proposed resolution, and the related issue 12.
 - Updated various sections of the spec to match the proposed resolution to issue 11.
- Revision 13, 2015-06-01 (James Jones)
 - Moved some structures to VK_EXT_KHR_swap_chain to resolve the specification's issues 1 and 2.
- Revision 14, 2015-06-01 (James Jones)
 - Added code for example 4 demonstrating how an application might make use of the two different present and acquire next image KHR result codes.
 - Added issue 13.
- Revision 15, 2015-06-01 (James Jones)
 - Added issues 14 - 16 and related spec language.
 - Fixed some spelling errors.
 - Added language describing the meaningful return values for vkAcquireNextImageKHR and

vkQueuePresentKHR.

- Revision 16, 2015-06-02 (James Jones)
 - Added issues 17 and 18, as well as related spec language.
 - Removed some erroneous text added by mistake in the last update.
- Revision 17, 2015-06-15 (Ian Elliott)
 - Changed special value from “-1” to “0” so that the data types can be unsigned.
- Revision 18, 2015-06-15 (Ian Elliott)
 - Clarified the values of `VkSurfacePropertiesKHR::minImageCount` and the timeout parameter of the `vkAcquireNextImageKHR` function.
- Revision 19, 2015-06-17 (James Jones)
 - Misc. cleanup. Removed resolved inline issues and fixed typos.
 - Fixed clarification of `VkSurfacePropertiesKHR::minImageCount` made in version 18.
 - Added a brief “Image Ownership” definition to the list of terms used in the spec.
- Revision 20, 2015-06-17 (James Jones)
 - Updated enum-extending values using new convention.
- Revision 21, 2015-06-17 (James Jones)
 - Added language describing how to use `VK_IMAGE_LAYOUT_PRESENT_SOURCE_KHR`.
 - Cleaned up an XXX comment regarding the description of which queues `vkQueuePresentKHR` can be used on.
- Revision 22, 2015-06-17 (James Jones)
 - Rebased on Vulkan API version 126.
- Revision 23, 2015-06-18 (James Jones)
 - Updated language for issue 12 to read as a proposed resolution.
 - Marked issues 11, 12, 13, 16, and 17 resolved.
 - Temporarily added links to the relevant bugs under the remaining unresolved issues.
 - Added issues 19 and 20 as well as proposed resolutions.
- Revision 24, 2015-06-19 (Ian Elliott)
 - Changed special value for `VkSurfacePropertiesKHR::currentExtent` back to “-1” from “0”. This value will never need to be unsigned, and “0” is actually a legal value.
- Revision 25, 2015-06-23 (Ian Elliott)
 - Examples now show use of function pointers for extension functions.
 - Eliminated extraneous whitespace.
- Revision 26, 2015-06-25 (Ian Elliott)
 - Resolved Issues 9 & 10 per KHR TSG meeting.
- Revision 27, 2015-06-25 (James Jones)

- Added oldSwapchain member to VkSwapchainCreateInfoKHR.
- Revision 28, 2015-06-25 (James Jones)
 - Added the “inherit” bits to the rotation and mirroring flags and the associated issue 21.
- Revision 29, 2015-06-25 (James Jones)
 - Added the “clipped” flag to VkSwapchainCreateInfoKHR, and the associated issue 22.
 - Specified that presenting an image does not modify it.
- Revision 30, 2015-06-25 (James Jones)
 - Added language to the spec that clarifies the behavior of vkCreateSwapchainKHR() when the oldSwapchain field of VkSwapchainCreateInfoKHR is not NULL.
- Revision 31, 2015-06-26 (Ian Elliott)
 - Example of new VkSwapchainCreateInfoKHR members, “oldSwapchain” and “clipped”.
 - Example of using VkSurfacePropertiesKHR::{min|max}ImageCount to set VkSwapchainCreateInfoKHR::minImageCount.
 - Rename vkGetSurfaceInfoKHR()'s 4th parameter to “pDataSize”, for consistency with other functions.
 - Add macro with C-string name of extension (just to header file).
- Revision 32, 2015-06-26 (James Jones)
 - Minor adjustments to the language describing the behavior of “oldSwapchain”
 - Fixed the version date on my previous two updates.
- Revision 33, 2015-06-26 (Jesse Hall)
 - Add usage flags to VkSwapchainCreateInfoKHR
- Revision 34, 2015-06-26 (Ian Elliott)
 - Rename vkQueuePresentKHR()'s 2nd parameter to “pPresentInfo”, for consistency with other functions.
- Revision 35, 2015-06-26 (Faith Ekstrand)
 - Merged the VkRotationFlagBitsKHR and VkMirrorFlagBitsKHR enums into a single VkSurfaceTransformFlagBitsKHR enum.
- Revision 36, 2015-06-26 (Faith Ekstrand)
 - Added a VkSurfaceTransformKHR enum that is not a bitmask. Each value in VkSurfaceTransformKHR corresponds directly to one of the bits in VkSurfaceTransformFlagBitsKHR so transforming from one to the other is easy. Having a separate enum means that currentTransform and preTransform are now unambiguous by definition.
- Revision 37, 2015-06-29 (Ian Elliott)
 - Corrected one of the signatures of vkAcquireNextImageKHR, which had the last two parameters switched from what it is elsewhere in the specification and header files.
- Revision 38, 2015-06-30 (Ian Elliott)

- Corrected a typo in description of the `vkGetSwapchainInfoKHR()` function.
- Corrected a typo in header file comment for `VkPresentInfoKHR::sType`.
- Revision 39, 2015-07-07 (Daniel Rakos)
 - Added error section describing when each error is expected to be reported.
 - Replaced `bool32_t` with `VkBool32`.
- Revision 40, 2015-07-10 (Ian Elliott)
 - Updated to work with version 138 of the `vulkan.h` header. This includes declaring the `VkSwapchainKHR` type using the new `VK_DEFINE_NONDISP_HANDLE` macro, and no longer extending `VkObjectType` (which was eliminated).
- Revision 41 2015-07-09 (Mathias Heyer)
 - Added color space language.
- Revision 42, 2015-07-10 (Daniel Rakos)
 - Updated query mechanism to reflect the convention changes done in the core spec.
 - Removed “queue” from the name of `VK_STRUCTURE_TYPE_QUEUE_PRESENT_INFO_KHR` to be consistent with the established naming convention.
 - Removed reference to the no longer existing `VkObjectType` enum.
- Revision 43, 2015-07-17 (Daniel Rakos)
 - Added support for concurrent sharing of swapchain images across queue families.
 - Updated sample code based on recent changes
- Revision 44, 2015-07-27 (Ian Elliott)
 - Noted that support for `VK_PRESENT_MODE_FIFO_KHR` is required. That is ICDs may optionally support `IMMEDIATE` and `MAILBOX`, but must support `FIFO`.
- Revision 45, 2015-08-07 (Ian Elliott)
 - Corrected a typo in spec file (type and variable name had wrong case for the `imageColorSpace` member of the `VkSwapchainCreateInfoKHR` struct).
 - Corrected a typo in header file (last parameter in `PFN_vkGetSurfacePropertiesKHR` was missing “KHR” at the end of type: `VkSurfacePropertiesKHR`).
- Revision 46, 2015-08-20 (Ian Elliott)
 - Renamed this extension and all of its enumerations, types, functions, etc. This makes it compliant with the proposed standard for Vulkan extensions.
 - Switched from “revision” to “version”, including use of the `VK_MAKE_VERSION` macro in the header file.
 - Made improvements to several descriptions.
 - Changed the status of several issues from `PROPOSED` to `RESOLVED`, leaving no unresolved issues.
 - Resolved several TODOs, did miscellaneous cleanup, etc.
- Revision 47, 2015-08-20 (Ian Elliott—porting a 2015-07-29 change from James Jones)

- Moved the surface transform enums to `VK_WSI_swapchain` so they could be reused by `VK_WSI_display`.
- Revision 48, 2015-09-01 (James Jones)
 - Various minor cleanups.
- Revision 49, 2015-09-01 (James Jones)
 - Restore single-field revision number.
- Revision 50, 2015-09-01 (James Jones)
 - Update Example #4 to include code that illustrates how to use the `oldSwapchain` field.
- Revision 51, 2015-09-01 (James Jones)
 - Fix example code compilation errors.
- Revision 52, 2015-09-08 (Matthaeus G. Chajdas)
 - Corrected a typo.
- Revision 53, 2015-09-10 (Alon Or-bach)
 - Removed underscore from `SWAP_CHAIN` left in `VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR`.
- Revision 54, 2015-09-11 (Jesse Hall)
 - Described the execution and memory coherence requirements for image transitions to and from `VK_IMAGE_LAYOUT_PRESENT_SOURCE_KHR`.
- Revision 55, 2015-09-11 (Ray Smith)
 - Added errors for destroying and binding memory to presentable images
- Revision 56, 2015-09-18 (James Jones)
 - Added fence argument to `vkAcquireNextImageKHR`
 - Added example of how to meter a host thread based on presentation rate.
- Revision 57, 2015-09-26 (Jesse Hall)
 - Replace `VkSurfaceDescriptionKHR` with `VkSurfaceKHR`.
 - Added issue 25 with agreed resolution.
- Revision 58, 2015-09-28 (Jesse Hall)
 - Renamed from `VK_EXT_KHR_device_swapchain` to `VK_EXT_KHR_swapchain`.
- Revision 59, 2015-09-29 (Ian Elliott)
 - Changed `vkDestroySwapchainKHR()` to return void.
- Revision 60, 2015-10-01 (Jeff Vigil)
 - Added error result `VK_ERROR_SURFACE_LOST_KHR`.
- Revision 61, 2015-10-05 (Faith Ekstrand)
 - Added the `VkCompositeAlpha` enum and corresponding structure fields.
- Revision 62, 2015-10-12 (Daniel Rakos)

- Added VK_PRESENT_MODE_FIFO_RELAXED_KHR.
- Revision 63, 2015-10-15 (Daniel Rakos)
 - Moved surface capability queries to VK_EXT_KHR_surface.
- Revision 64, 2015-10-26 (Ian Elliott)
 - Renamed from VK_EXT_KHR_swapchain to VK_KHR_swapchain.
- Revision 65, 2015-10-28 (Ian Elliott)
 - Added optional pResult member to VkPresentInfoKHR, so that per-swapchain results can be obtained from vkQueuePresentKHR().
- Revision 66, 2015-11-03 (Daniel Rakos)
 - Added allocation callbacks to create and destroy functions.
 - Updated resource transition language.
 - Updated sample code.
- Revision 67, 2015-11-10 (Jesse Hall)
 - Add reserved flags bitmask to VkSwapchainCreateInfoKHR.
 - Modify naming and member ordering to match API style conventions, and so the VkSwapchainCreateInfoKHR image property members mirror corresponding VkImageCreateInfo members but with an 'image' prefix.
 - Make VkPresentInfoKHR::pResults non-const; it is an output array parameter.
 - Make pPresentInfo parameter to vkQueuePresentKHR const.
- Revision 68, 2016-04-05 (Ian Elliott)
 - Moved the “validity” include for vkAcquireNextImage to be in its proper place, after the prototype and list of parameters.
 - Clarified language about presentable images, including how they are acquired, when applications can and cannot use them, etc. As part of this, removed language about “ownership” of presentable images, and replaced it with more-consistent language about presentable images being “acquired” by the application.
- 2016-08-23 (Ian Elliott)
 - Update the example code, to use the final API command names, to not have so many characters per line, and to split out a new example to show how to obtain function pointers. This code is more similar to the LunarG “cube” demo program.
- 2016-08-25 (Ian Elliott)
 - A note was added at the beginning of the example code, stating that it will be removed from future versions of the appendix.
- Revision 69, 2017-09-07 (Tobias Hector)
 - Added interactions with Vulkan 1.1
- Revision 70, 2017-10-06 (Ian Elliott)
 - Corrected interactions with Vulkan 1.1

VK_KHR_swapchain_mutable_format

Name String

VK_KHR_swapchain_mutable_format

Extension Type

Device extension

Registered Extension Number

201

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_swapchain](#)

and

[VK_KHR_maintenance2](#)

or

[Vulkan Version 1.1](#)

and

[VK_KHR_image_format_list](#)

or

[Vulkan Version 1.2](#)

Contact

- Daniel Rakos [\[GitHub\]](#)[drakos-amd](#)

Other Extension Metadata

Last Modified Date

2018-03-28

IP Status

No known IP claims.

Contributors

- Faith Ekstrand, Intel
- Jan-Harald Fredriksen, ARM
- Jesse Hall, Google
- Daniel Rakos, AMD
- Ray Smith, ARM

Description

This extension allows processing of swapchain images as different formats to that used by the window system, which is particularly useful for switching between sRGB and linear RGB formats.

It adds a new swapchain creation flag that enables creating image views from presentable images with a different format than the one used to create the swapchain.

New Enum Constants

- `VK_KHR_SWAPCHAIN_MUTABLE_FORMAT_EXTENSION_NAME`
- `VK_KHR_SWAPCHAIN_MUTABLE_FORMAT_SPEC_VERSION`
- Extending `VkSwapchainCreateFlagBitsKHR`:
 - `VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR`

Issues

1) Are there any new capabilities needed?

RESOLVED: No. It is expected that all implementations exposing this extension support swapchain image format mutability.

2) Do we need a separate `VK_SWAPCHAIN_CREATE_EXTENDED_USAGE_BIT_KHR`?

RESOLVED: No. This extension requires `VK_KHR_maintenance2` and presentable images of swapchains created with `VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR` are created internally in a way equivalent to specifying both `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` and `VK_IMAGE_CREATE_EXTENDED_USAGE_BIT_KHR`.

3) Do we need a separate structure to allow specifying an image format list for swapchains?

RESOLVED: No. We simply use the same `VkImageFormatListCreateInfoKHR` structure introduced by `VK_KHR_image_format_list`. The structure is required to be included in the `pNext` chain of `VkSwapchainCreateInfoKHR` for swapchains created with `VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR`.

Version History

- Revision 1, 2018-03-28 (Daniel Rakos)
 - Internal revisions.

VK_KHR_synchronization2

Name String

`VK_KHR_synchronization2`

Extension Type

Device extension

Registered Extension Number

315

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

Vulkan Version 1.1

API Interactions

- Interacts with VK_EXT_blend_operation_advanced
- Interacts with VK_EXT_conditional_rendering
- Interacts with VK_EXT_device_generated_commands
- Interacts with VK_EXT_fragment_density_map
- Interacts with VK_EXT_mesh_shader
- Interacts with VK_EXT_transform_feedback
- Interacts with VK_KHR_acceleration_structure
- Interacts with VK_KHR_fragment_shading_rate
- Interacts with VK_KHR_ray_tracing_pipeline
- Interacts with VK_NV_device_generated_commands
- Interacts with VK_NV_mesh_shader
- Interacts with VK_NV_ray_tracing
- Interacts with VK_NV_shading_rate_image

Deprecation State

- *Promoted* to Vulkan 1.3

Contact

- Tobias Hector [\[GitHub\]tobski](#)

Other Extension Metadata

Last Modified Date

2020-12-03

Interactions and External Dependencies

- Interacts with VK_KHR_create_renderpass2

Contributors

- Tobias Hector

Description

This extension modifies the original core synchronization APIs to simplify the interface and improve usability of these APIs. It also adds new pipeline stage and access flag types that extend into the 64-bit range, as we have run out within the 32-bit range. The new flags are identical to the old values within the 32-bit range, with new stages and bits beyond that.

Pipeline stages and access flags are now specified together in memory barrier structures, making the connection between the two more obvious. Additionally, scoping the pipeline stages into the barrier structs allows the use of the `MEMORY_READ` and `MEMORY_WRITE` flags without sacrificing precision. The per-stage access flags should be used to disambiguate specific accesses in a given stage or set of stages - for instance, between uniform reads and sampling operations.

Layout transitions have been simplified as well; rather than requiring a different set of layouts for depth/stencil/color attachments, there are generic `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR` and `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR` layouts which are contextually applied based on the image format. For example, for a depth format image, `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR` is equivalent to `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL_KHR`. `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR` also functionally replaces `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`.

Events are now more efficient, because they include memory dependency information when you set them on the device. Previously, this information was only known when waiting on an event, so the dependencies could not be satisfied until the wait occurred. That sometimes meant stalling the pipeline when the wait occurred. The new API provides enough information for implementations to satisfy these dependencies in parallel with other tasks.

Queue submission has been changed to wrap command buffers and semaphores in extensible structures, which incorporate changes from Vulkan 1.1, `VK_KHR_device_group`, and `VK_KHR_timeline_semaphore`. This also adds a pipeline stage to the semaphore signal operation, mirroring the existing pipeline stage specification for wait operations.

Other miscellaneous changes include:

- Events can now be specified as interacting only with the device, allowing more efficient access to the underlying object.
- Image memory barriers that do not perform an image layout transition can be specified by setting `oldLayout` equal to `newLayout`.
 - E.g. the old and new layout can both be set to `VK_IMAGE_LAYOUT_UNDEFINED`, without discarding data in the image.
- Queue family ownership transfer parameters are simplified in some cases.
- Extensions with commands or functions with a `VkPipelineStageFlags` or `VkPipelineStageFlagBits` parameter have had those APIs replaced with equivalents using `VkPipelineStageFlags2KHR`.
- The new event and barrier interfaces are now more extensible for future changes.
- Relevant pipeline stage masks can now be specified as empty with the new

[VK_PIPELINE_STAGE_NONE_KHR](#) and [VK_PIPELINE_STAGE_2_NONE_KHR](#) values.

- [VkMemoryBarrier2KHR](#) can be chained to [VkSubpassDependency2](#), overriding the original 32-bit stage and access masks.

New Base Types

- [VkFlags64](#)

New Commands

- [vkCmdPipelineBarrier2KHR](#)
- [vkCmdResetEvent2KHR](#)
- [vkCmdSetEvent2KHR](#)
- [vkCmdWaitEvents2KHR](#)
- [vkCmdWriteTimestamp2KHR](#)
- [vkQueueSubmit2KHR](#)

New Structures

- [VkBufferMemoryBarrier2KHR](#)
- [VkCommandBufferSubmitInfoKHR](#)
- [VkDependencyInfoKHR](#)
- [VkImageMemoryBarrier2KHR](#)
- [VkSemaphoreSubmitInfoKHR](#)
- [VkSubmitInfo2KHR](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceSynchronization2FeaturesKHR](#)
- Extending [VkSubpassDependency2](#):
 - [VkMemoryBarrier2KHR](#)

New Enums

- [VkAccessFlagBits2KHR](#)
- [VkPipelineStageFlagBits2KHR](#)
- [VkSubmitFlagBitsKHR](#)

New Bitmasks

- [VkAccessFlags2KHR](#)
- [VkPipelineStageFlags2KHR](#)
- [VkSubmitFlagsKHR](#)

New Enum Constants

- `VK_KHR_SYNCHRONIZATION_2_EXTENSION_NAME`
- `VK_KHR_SYNCHRONIZATION_2_SPEC_VERSION`
- Extending `VkAccessFlagBits`:
 - `VK_ACCESS_NONE_KHR`
- Extending `VkAccessFlagBits2`:
 - `VK_ACCESS_2_COLOR_ATTACHMENT_READ_BIT_KHR`
 - `VK_ACCESS_2_COLOR_ATTACHMENT_WRITE_BIT_KHR`
 - `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_READ_BIT_KHR`
 - `VK_ACCESS_2_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT_KHR`
 - `VK_ACCESS_2_HOST_READ_BIT_KHR`
 - `VK_ACCESS_2_HOST_WRITE_BIT_KHR`
 - `VK_ACCESS_2_INDEX_READ_BIT_KHR`
 - `VK_ACCESS_2_INDIRECT_COMMAND_READ_BIT_KHR`
 - `VK_ACCESS_2_INPUT_ATTACHMENT_READ_BIT_KHR`
 - `VK_ACCESS_2_MEMORY_READ_BIT_KHR`
 - `VK_ACCESS_2_MEMORY_WRITE_BIT_KHR`
 - `VK_ACCESS_2_NONE_KHR`
 - `VK_ACCESS_2_SHADER_READ_BIT_KHR`
 - `VK_ACCESS_2_SHADER_SAMPLED_READ_BIT_KHR`
 - `VK_ACCESS_2_SHADER_STORAGE_READ_BIT_KHR`
 - `VK_ACCESS_2_SHADER_STORAGE_WRITE_BIT_KHR`
 - `VK_ACCESS_2_SHADER_WRITE_BIT_KHR`
 - `VK_ACCESS_2_TRANSFER_READ_BIT_KHR`
 - `VK_ACCESS_2_TRANSFER_WRITE_BIT_KHR`
 - `VK_ACCESS_2_UNIFORM_READ_BIT_KHR`
 - `VK_ACCESS_2_VERTEX_ATTRIBUTE_READ_BIT_KHR`
- Extending `VkEventCreateFlagBits`:
 - `VK_EVENT_CREATE_DEVICE_ONLY_BIT_KHR`
- Extending `VkImageLayout`:
 - `VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR`
 - `VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR`
- Extending `VkPipelineStageFlagBits`:
 - `VK_PIPELINE_STAGE_NONE_KHR`

- Extending [VkPipelineStageFlagBits2](#):

- VK_PIPELINE_STAGE_2_ALL_COMMANDS_BIT_KHR
- VK_PIPELINE_STAGE_2_ALL_GRAPHICS_BIT_KHR
- VK_PIPELINE_STAGE_2_ALL_TRANSFER_BIT_KHR
- VK_PIPELINE_STAGE_2_BLIT_BIT_KHR
- VK_PIPELINE_STAGE_2_BOTTOM_OF_PIPE_BIT_KHR
- VK_PIPELINE_STAGE_2_CLEAR_BIT_KHR
- VK_PIPELINE_STAGE_2_COLOR_ATTACHMENT_OUTPUT_BIT_KHR
- VK_PIPELINE_STAGE_2_COMPUTE_SHADER_BIT_KHR
- VK_PIPELINE_STAGE_2_COPY_BIT_KHR
- VK_PIPELINE_STAGE_2_DRAW_INDIRECT_BIT_KHR
- VK_PIPELINE_STAGE_2_EARLY_FRAGMENT_TESTS_BIT_KHR
- VK_PIPELINE_STAGE_2_FRAGMENT_SHADER_BIT_KHR
- VK_PIPELINE_STAGE_2_GEOMETRY_SHADER_BIT_KHR
- VK_PIPELINE_STAGE_2_HOST_BIT_KHR
- VK_PIPELINE_STAGE_2_INDEX_INPUT_BIT_KHR
- VK_PIPELINE_STAGE_2_LATE_FRAGMENT_TESTS_BIT_KHR
- VK_PIPELINE_STAGE_2_NONE_KHR
- VK_PIPELINE_STAGE_2_PRE_RASTERIZATION_SHADERS_BIT_KHR
- VK_PIPELINE_STAGE_2_RESOLVE_BIT_KHR
- VK_PIPELINE_STAGE_2_TESSELLATION_CONTROL_SHADER_BIT_KHR
- VK_PIPELINE_STAGE_2_TESSELLATION_EVALUATION_SHADER_BIT_KHR
- VK_PIPELINE_STAGE_2_TOP_OF_PIPE_BIT_KHR
- VK_PIPELINE_STAGE_2_TRANSFER_BIT_KHR
- VK_PIPELINE_STAGE_2_VERTEX_ATTRIBUTE_INPUT_BIT_KHR
- VK_PIPELINE_STAGE_2_VERTEX_INPUT_BIT_KHR
- VK_PIPELINE_STAGE_2_VERTEX_SHADER_BIT_KHR

- Extending [VkStructureType](#):

- VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER_2_KHR
- VK_STRUCTURE_TYPE_COMMAND_BUFFER_SUBMIT_INFO_KHR
- VK_STRUCTURE_TYPE_DEPENDENCY_INFO_KHR
- VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER_2_KHR
- VK_STRUCTURE_TYPE_MEMORY_BARRIER_2_KHR
- VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SYNCHRONIZATION_2_FEATURES_KHR
- VK_STRUCTURE_TYPE_SEMAPHORE_SUBMIT_INFO_KHR

- VK_STRUCTURE_TYPE_SUBMIT_INFO_2_KHR
- Extending VkSubmitFlagBits:
 - VK_SUBMIT_PROTECTED_BIT_KHR

If VK_EXT_blend_operation_advanced is supported:

- Extending VkAccessFlagBits2:
 - VK_ACCESS_2_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT

If VK_EXT_conditional_rendering is supported:

- Extending VkAccessFlagBits2:
 - VK_ACCESS_2_CONDITIONAL_RENDERING_READ_BIT_EXT
- Extending VkPipelineStageFlagBits2:
 - VK_PIPELINE_STAGE_2_CONDITIONAL_RENDERING_BIT_EXT

If VK_EXT_device_generated_commands is supported:

- Extending VkAccessFlagBits2:
 - VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_EXT
 - VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_EXT
- Extending VkPipelineStageFlagBits2:
 - VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_EXT

If VK_EXT_fragment_density_map is supported:

- Extending VkAccessFlagBits2:
 - VK_ACCESS_2_FRAGMENT_DENSITY_MAP_READ_BIT_EXT
- Extending VkPipelineStageFlagBits2:
 - VK_PIPELINE_STAGE_2_FRAGMENT_DENSITY_PROCESS_BIT_EXT

If VK_EXT_mesh_shader is supported:

- Extending VkPipelineStageFlagBits2:
 - VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_EXT
 - VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_EXT

If VK_EXT_transform_feedback is supported:

- Extending VkAccessFlagBits2:
 - VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT
 - VK_ACCESS_2_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT
 - VK_ACCESS_2_TRANSFORM_FEEDBACK_WRITE_BIT_EXT

- Extending [VkPipelineStageFlagBits2](#):
 - `VK_PIPELINE_STAGE_2_TRANSFORM_FEEDBACK_BIT_EXT`

If `VK_KHR_acceleration_structure` is supported:

- Extending [VkAccessFlagBits2](#):
 - `VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_KHR`
 - `VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_KHR`
- Extending [VkPipelineStageFlagBits2](#):
 - `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_KHR`

If `VK_KHR_fragment_shading_rate` is supported:

- Extending [VkAccessFlagBits2](#):
 - `VK_ACCESS_2_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR`
- Extending [VkPipelineStageFlagBits2](#):
 - `VK_PIPELINE_STAGE_2_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR`

If `VK_KHR_ray_tracing_pipeline` is supported:

- Extending [VkPipelineStageFlagBits2](#):
 - `VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_KHR`

If `VK_NV_device_generated_commands` is supported:

- Extending [VkAccessFlagBits2](#):
 - `VK_ACCESS_2_COMMAND_PREPROCESS_READ_BIT_NV`
 - `VK_ACCESS_2_COMMAND_PREPROCESS_WRITE_BIT_NV`
- Extending [VkPipelineStageFlagBits2](#):
 - `VK_PIPELINE_STAGE_2_COMMAND_PREPROCESS_BIT_NV`

If `VK_NV_mesh_shader` is supported:

- Extending [VkPipelineStageFlagBits2](#):
 - `VK_PIPELINE_STAGE_2_MESH_SHADER_BIT_NV`
 - `VK_PIPELINE_STAGE_2_TASK_SHADER_BIT_NV`

If `VK_NV_ray_tracing` is supported:

- Extending [VkAccessFlagBits2](#):
 - `VK_ACCESS_2_ACCELERATION_STRUCTURE_READ_BIT_NV`
 - `VK_ACCESS_2_ACCELERATION_STRUCTURE_WRITE_BIT_NV`
- Extending [VkPipelineStageFlagBits2](#):

- `VK_PIPELINE_STAGE_2_ACCELERATION_STRUCTURE_BUILD_BIT_NV`
- `VK_PIPELINE_STAGE_2_RAY_TRACING_SHADER_BIT_NV`

If `VK_NV_shading_rate_image` is supported:

- Extending `VkAccessFlagBits2`:
 - `VK_ACCESS_2_SHADING_RATE_IMAGE_READ_BIT_NV`
- Extending `VkPipelineStageFlagBits2`:
 - `VK_PIPELINE_STAGE_2_SHADING_RATE_IMAGE_BIT_NV`

Promotion to Vulkan 1.3

Vulkan APIs in this extension are included in core Vulkan 1.3, with the KHR suffix omitted. External interactions defined by this extension, such as SPIR-V token names, retain their original names. The original Vulkan API names are still available as aliases of the core functionality.

Examples

See <https://github.com/KhronosGroup/Vulkan-Docs/wiki/Synchronization-Examples>

Version History

- Revision 1, 2020-12-03 (Tobias Hector)
 - Internal revisions

VK_KHR_vertex_attribute_divisor

Name String

`VK_KHR_vertex_attribute_divisor`

Extension Type

Device extension

Registered Extension Number

526

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_get_physical_device_properties2`

or

[Vulkan Version 1.1](#)

Deprecation State

- *Promoted* to Vulkan 1.4

Contact

- Shahbaz Youssefi [\[GitHub\]syoussefi](#)

Extension Proposal

[VK_KHR_vertex_attribute_divisor](#)

Other Extension Metadata

Last Modified Date

2023-09-20

IP Status

No known IP claims.

Contributors

- Shahbaz Youssefi, Google
- Contributors to [VK_EXT_vertex_attribute_divisor](#)

Description

This extension is based on the [VK_EXT_vertex_attribute_divisor](#) extension. The only difference is the new property [supportsNonZeroFirstInstance](#), which indicates support for non-zero values in [firstInstance](#). This allows the extension to be supported on implementations that have traditionally only supported OpenGL ES.

New Structures

- [VkVertexInputBindingDivisorDescriptionKHR](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceVertexAttributeDivisorFeaturesKHR](#)
- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDeviceVertexAttributeDivisorPropertiesKHR](#)
- Extending [VkPipelineVertexInputStateCreateInfo](#):
 - [VkPipelineVertexInputDivisorStateCreateInfoKHR](#)

New Enum Constants

- [VK_KHR_VERTEX_ATTRIBUTE_DIVISOR_EXTENSION_NAME](#)
- [VK_KHR_VERTEX_ATTRIBUTE_DIVISOR_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_FEATURES_KHR](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_ATTRIBUTE_DIVISOR_PROPERTIES_KHR](#)

- `VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_DIVISOR_STATE_CREATE_INFO_KHR`

Promotion to Vulkan 1.4

Functionality in this extension is included in core Vulkan 1.4 with the KHR suffix omitted. The original type, enum and command names are still available as aliases of the core functionality.

Version History

- Revision 1, 2023-09-20 (Shahbaz Youssefi)
 - First Version, based on `VK_EXT_vertex_attribute_divisor`

VK_EXT_4444_formats

Name String

`VK_EXT_4444_formats`

Extension Type

Device extension

Registered Extension Number

341

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_get_physical_device_properties2`

or

[Vulkan Version 1.1](#)

Deprecation State

- *Promoted* to Vulkan 1.3

Contact

- Joshua Ashton [\[GitHub\]](#)[Joshua-Ashton](#)

Other Extension Metadata

Last Modified Date

2020-07-28

IP Status

No known IP claims.

Contributors

- Joshua Ashton, Valve
- Faith Ekstrand, Intel

Description

This extension defines the `VK_FORMAT_A4R4G4B4_UNORM_PACK16_EXT` and `VK_FORMAT_A4B4G4R4_UNORM_PACK16_EXT` formats which are defined in other current graphics APIs.

This extension may be useful for building translation layers for those APIs or for porting applications that use these formats without having to resort to swizzles.

When `VK_EXT_custom_border_color` is used, these formats are not subject to the same restrictions for border color without format as with `VK_FORMAT_B4G4R4A4_UNORM_PACK16`.

New Structures

- Extending `VkPhysicalDeviceFeatures2`, `VkDeviceCreateInfo`:
 - `VkPhysicalDevice4444FormatsFeaturesEXT`

New Enum Constants

- `VK_EXT_4444_FORMATS_EXTENSION_NAME`
- `VK_EXT_4444_FORMATS_SPEC_VERSION`
- Extending `VkFormat`:
 - `VK_FORMAT_A4B4G4R4_UNORM_PACK16_EXT`
 - `VK_FORMAT_A4R4G4B4_UNORM_PACK16_EXT`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_4444_FORMATS_FEATURES_EXT`

Promotion to Vulkan 1.3

The format enumerants introduced by the extension are included in core Vulkan 1.3, with the EXT suffix omitted. However, runtime support for these formats is optional in core Vulkan 1.3, while if this extension is supported, runtime support is mandatory. The feature structure is not promoted. The original enum names are still available as aliases of the core functionality.

Version History

- Revision 1, 2020-07-04 (Joshua Ashton)
 - Initial draft

VK_EXT_application_parameters

Name String

`VK_EXT_application_parameters`

Extension Type

Instance extension

Registered Extension Number

436

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

None

Contact

- Daniel Koch [\[GitHub\]dgkoch](#)

Other Extension Metadata

Last Modified Date

2021-12-14

Contributors

- Daniel Koch, NVIDIA
- Jonathan Mccaffrey, NVIDIA
- Aidan Fabius, CoreAVI

Description

This instance extension enables an application to pass application parameters to the implementation at instance or device creation time.

The application parameters consist of a set of vendor-specific keys and values. Each key is a 32-bit enum, and each value is a 64-bit integer. The valid keys, range of values, and default values are documented external to this specification in implementation-specific documentation.

This extension is an instance extension rather than a device extension so that the implementation can modify reported `VkPhysicalDevice` properties or features as needed.

New Structures

- Extending [VkApplicationInfo](#), [VkDeviceCreateInfo](#):
 - [VkApplicationParametersEXT](#)

New Enum Constants

- `VK_EXT_APPLICATION_PARAMETERS_EXTENSION_NAME`

- `VK_EXT_APPLICATION_PARAMETERS_SPEC_VERSION`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_APPLICATION_PARAMETERS_EXT`

Issues

1. How should the `key` enumerants be assigned?

RESOLVED: The `key` enumerants are completely implementation-specific and do not need to be centrally reserved. They should be documented in the implementation-specific documentation. The vendor ID and optionally the device ID are provided to disambiguate between multiple ICDs or devices.

2. How does an application know what application parameters are valid on a particular implementation?

DISCUSSION: There is no ability to enumerate device or system properties before an instance is created, however `key` and `values` **must** be recognized by an implementation in order for instance or device creation to succeed. The vendor and optionally the device ID are provided to identify which ICD or device the application parameters are targeted at.

3. Is it OK if the "valid value" for specified keys is not from static documented values, but must be consistent-with/interdependent-on other `VkApplicationParametersEXT`?

DISCUSSION: Yes this is fine. Examples for how this could be used include:

- a checksum `key` where the `value` is computed based on other `VkApplicationParametersEXT` structures in the `pNext` chain.
- an "application key" which either implies or explicitly lists a set of prevalidated key/value pairs.

Version History

- Revision 1, 2021-12-14 (Daniel Koch)
 - Initial revision

VK_EXT_astc_decode_mode

Name String

`VK_EXT_astc_decode_mode`

Extension Type

Device extension

Registered Extension Number

68

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_get_physical_device_properties2](#)

or

[Vulkan Version 1.1](#)

Contact

- Jan-Harald Fredriksen [\[GitHub\]janharaldfredriksen-arm](#)

Other Extension Metadata

Last Modified Date

2018-08-07

Contributors

- Jan-Harald Fredriksen, Arm

Description

The existing specification requires that low dynamic range (LDR) ASTC textures are decompressed to FP16 values per component. In many cases, decompressing LDR textures to a lower precision intermediate result gives acceptable image quality. Source material for LDR textures is typically authored as 8-bit UNORM values, so decoding to FP16 values adds little value. On the other hand, reducing precision of the decoded result reduces the size of the decompressed data, potentially improving texture cache performance and saving power.

The goal of this extension is to enable this efficiency gain on existing ASTC texture data. This is achieved by giving the application the ability to select the intermediate decoding precision.

Three decoding options are provided:

- Decode to [VK_FORMAT_R16G16B16A16_SFLOAT](#) precision: This is the default, and matches the required behavior in the core API.
- Decode to [VK_FORMAT_R8G8B8A8_UNORM](#) precision: This is provided as an option in LDR mode.
- Decode to [VK_FORMAT_E5B9G9R9_UFLOAT_PACK32](#) precision: This is provided as an option in both LDR and HDR mode. In this mode, negative values cannot be represented and are clamped to zero. The alpha component is ignored, and the results are as if alpha was 1.0. This decode mode is optional and support can be queried via the physical device properties.

New Structures

- Extending [VkImageViewCreateInfo](#):
 - [VkImageViewASTCDecodeModeEXT](#)

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceASTCDecodeFeaturesEXT](#)

New Enum Constants

- [VK_EXT_ASTC_DECODE_MODE_EXTENSION_NAME](#)
- [VK_EXT_ASTC_DECODE_MODE_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_IMAGE_VIEW_ASTC_DECODE_MODE_EXT](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ASTC_DECODE_FEATURES_EXT](#)

Issues

1) Are implementations allowed to decode at a higher precision than what is requested?

RESOLUTION: No.

If we allow this, then this extension could be exposed on all implementations that support ASTC.

But developers would have no way of knowing what precision was actually used, and thus whether the image quality is sufficient at reduced precision.

2) Should the decode mode be image view state and/or sampler state?

RESOLUTION: Image view state only.

Some implementations treat the different decode modes as different texture formats.

Example

Create an image view that decodes to [VK_FORMAT_R8G8B8A8_UNORM](#) precision:

```
VkImageViewASTCDecodeModeEXT decodeMode =
{
    .sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_ASTC_DECODE_MODE_EXT,
    .pNext = NULL,
    .decodeMode = VK_FORMAT_R8G8B8A8_UNORM
};

VkImageViewCreateInfo createInfo =
{
    .sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO,
    .pNext = &decodeMode,
    // flags, image, viewType set to application-desired values
    .format = VK_FORMAT_ASTC_8x8_UNORM_BLOCK,
    // components, subresourceRange set to application-desired values
}
```

```
};

VkImageView imageView;
VkResult result = vkCreateImageView(
    device,
    &createInfo,
    NULL,
    &imageView);
```

Version History

- Revision 1, 2018-08-07 (Jan-Harald Fredriksen)
 - Initial revision

VK_EXT_blend_operation_advanced

Name String

VK_EXT_blend_operation_advanced

Extension Type

Device extension

Registered Extension Number

149

Revision

2

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

[Vulkan Version 1.1](#)

Contact

- Jeff Bolz [\[GitHub\]jeffbolznv](#)

Other Extension Metadata

Last Modified Date

2017-06-12

Contributors

- Jeff Bolz, NVIDIA

Description

This extension adds a number of “advanced” blending operations that **can** be used to perform new color blending operations, many of which are more complex than the standard blend modes provided by unextended Vulkan. This extension requires different styles of usage, depending on the level of hardware support and the enabled features:

- If `VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT::advancedBlendCoherentOperations` is `VK_FALSE`, the new blending operations are supported, but a memory dependency **must** separate each advanced blend operation on a given sample. `VK_ACCESS_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT` is used to synchronize reads using advanced blend operations.
- If `VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT::advancedBlendCoherentOperations` is `VK_TRUE`, advanced blend operations obey primitive order just like basic blend operations.

In unextended Vulkan, the set of blending operations is limited, and **can** be expressed very simply. The `VK_BLEND_OP_MIN` and `VK_BLEND_OP_MAX` blend operations simply compute component-wise minimums or maximums of source and destination color components. The `VK_BLEND_OP_ADD`, `VK_BLEND_OP_SUBTRACT`, and `VK_BLEND_OP_REVERSE_SUBTRACT` modes multiply the source and destination colors by source and destination factors and either add the two products together or subtract one from the other. This limited set of operations supports many common blending operations but precludes the use of more sophisticated transparency and blending operations commonly available in many dedicated imaging APIs.

This extension provides a number of new “advanced” blending operations. Unlike traditional blending operations using `VK_BLEND_OP_ADD`, these blending equations do not use source and destination factors specified by `VkBlendFactor`. Instead, each blend operation specifies a complete equation based on the source and destination colors. These new blend operations are used for both RGB and alpha components; they **must** not be used to perform separate RGB and alpha blending (via different values of color and alpha `VkBlendOp`).

These blending operations are performed using premultiplied colors, where RGB colors **can** be considered premultiplied or non-premultiplied by alpha, according to the `srcPremultiplied` and `dstPremultiplied` members of `VkPipelineColorBlendAdvancedStateCreateInfoEXT`. If a color is considered non-premultiplied, the (R,G,B) color components are multiplied by the alpha component prior to blending. For non-premultiplied color components in the range [0,1], the corresponding premultiplied color component would have values in the range $[0 \times A, 1 \times A]$.

Many of these advanced blending equations are formulated where the result of blending source and destination colors with partial coverage have three separate contributions: from the portions covered by both the source and the destination, from the portion covered only by the source, and from the portion covered only by the destination. The blend parameter `VkPipelineColorBlendAdvancedStateCreateInfoEXT::blendOverlap` **can** be used to specify a correlation between source and destination pixel coverage. If set to `VK_BLEND_OVERLAP_CONJOINT_EXT`, the source and destination are considered to have maximal overlap, as would be the case if drawing two objects on top of each other. If set to `VK_BLEND_OVERLAP_DISJOINT_EXT`, the source and destination are considered to have minimal overlap, as would be the case when rendering a complex polygon tessellated into individual non-intersecting triangles. If set to `VK_BLEND_OVERLAP_UNCORRELATED_EXT`, the source and destination coverage are assumed to have no

spatial correlation within the pixel.

In addition to the coherency issues on implementations not supporting [advancedBlendCoherentOperations](#), this extension has several limitations worth noting. First, the new blend operations have a limit on the number of color attachments they **can** be used with, as indicated by [VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT::advancedBlendMaxColorAttachments](#). Additionally, blending precision **may** be limited to 16-bit floating-point, which **may** result in a loss of precision and dynamic range for framebuffer formats with 32-bit floating-point components, and in a loss of precision for formats with 12- and 16-bit signed or unsigned normalized integer components.

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT](#)
- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDeviceBlendOperationAdvancedPropertiesEXT](#)
- Extending [VkPipelineColorBlendStateCreateInfo](#):
 - [VkPipelineColorBlendAdvancedStateCreateInfoEXT](#)

New Enums

- [VkBlendOverlapEXT](#)

New Enum Constants

- [VK_EXT_BLEND_OPERATION_ADVANCED_EXTENSION_NAME](#)
- [VK_EXT_BLEND_OPERATION_ADVANCED_SPEC_VERSION](#)
- Extending [VkAccessFlagBits](#):
 - [VK_ACCESS_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT](#)
- Extending [VkBlendOp](#):
 - [VK_BLEND_OP_BLUE_EXT](#)
 - [VK_BLEND_OP_COLORBURN_EXT](#)
 - [VK_BLEND_OP_COLORDODGE_EXT](#)
 - [VK_BLEND_OP_CONTRAST_EXT](#)
 - [VK_BLEND_OP_DARKEN_EXT](#)
 - [VK_BLEND_OP_DIFFERENCE_EXT](#)
 - [VK_BLEND_OP_DST_ATOP_EXT](#)
 - [VK_BLEND_OP_DST_EXT](#)
 - [VK_BLEND_OP_DST_IN_EXT](#)
 - [VK_BLEND_OP_DST_OUT_EXT](#)

- VK_BLEND_OP_DST_OVER_EXT
- VK_BLEND_OP_EXCLUSION_EXT
- VK_BLEND_OP_GREEN_EXT
- VK_BLEND_OP_HARDLIGHT_EXT
- VK_BLEND_OP_HARDMIX_EXT
- VK_BLEND_OP_HSL_COLOR_EXT
- VK_BLEND_OP_HSL_HUE_EXT
- VK_BLEND_OP_HSL_LUMINOSITY_EXT
- VK_BLEND_OP_HSL_SATURATION_EXT
- VK_BLEND_OP_INVERT_EXT
- VK_BLEND_OP_INVERT_OVG_EXT
- VK_BLEND_OP_INVERT_RGB_EXT
- VK_BLEND_OP_LIGHTEN_EXT
- VK_BLEND_OP_LINEARBURN_EXT
- VK_BLEND_OP_LINEARDODGE_EXT
- VK_BLEND_OP_LINEARLIGHT_EXT
- VK_BLEND_OP_MINUS_CLAMPED_EXT
- VK_BLEND_OP_MINUS_EXT
- VK_BLEND_OP_MULTIPLY_EXT
- VK_BLEND_OP_OVERLAY_EXT
- VK_BLEND_OP_PINLIGHT_EXT
- VK_BLEND_OP_PLUS_CLAMPED_ALPHA_EXT
- VK_BLEND_OP_PLUS_CLAMPED_EXT
- VK_BLEND_OP_PLUS_DARKER_EXT
- VK_BLEND_OP_PLUS_EXT
- VK_BLEND_OP_RED_EXT
- VK_BLEND_OP_SCREEN_EXT
- VK_BLEND_OP_SOFTLIGHT_EXT
- VK_BLEND_OP_SRC_ATOP_EXT
- VK_BLEND_OP_SRC_EXT
- VK_BLEND_OP_SRC_IN_EXT
- VK_BLEND_OP_SRC_OUT_EXT
- VK_BLEND_OP_SRC_OVER_EXT
- VK_BLEND_OP_VIVIDLIGHT_EXT
- VK_BLEND_OP_XOR_EXT

- `VK_BLEND_OP_ZERO_EXT`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BLEND_OPERATION_ADVANCED_FEATURES_EXT`
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_BLEND_OPERATION_ADVANCED_PROPERTIES_EXT`
 - `VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_ADVANCED_STATE_CREATE_INFO_EXT`

Issues

None.

Version History

- Revision 1, 2017-06-12 (Jeff Bolz)
 - Internal revisions
- Revision 2, 2017-06-12 (Jeff Bolz)
 - Internal revisions

VK_EXT_color_write_enable

Name String

`VK_EXT_color_write_enable`

Extension Type

Device extension

Registered Extension Number

382

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_get_physical_device_properties2`
or
[Vulkan Version 1.1](#)

Contact

- Sharif Elcott [\[GitHub\]](#)[selcott](#)

Other Extension Metadata

Last Modified Date

2020-02-25

IP Status

No known IP claims.

Contributors

- Sharif Elcott, Google
- Tobias Hector, AMD
- Piers Daniell, NVIDIA

Description

This extension allows for selectively enabling and disabling writes to output color attachments via a pipeline dynamic state.

The intended use cases for this new state are mostly identical to those of `colorWriteMask`, such as selectively disabling writes to avoid feedback loops between subpasses or bandwidth savings for unused outputs. By making the state dynamic, one additional benefit is the ability to reduce pipeline counts and pipeline switching via shaders that write a superset of the desired data of which subsets are selected dynamically. The reason for a new state, `colorWriteEnable`, rather than making `colorWriteMask` dynamic is that, on many implementations, the more flexible per-component semantics of the `colorWriteMask` state cannot be made dynamic in a performant manner.

New Commands

- [vkCmdSetColorWriteEnableEXT](#)

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceColorWriteEnableFeaturesEXT](#)
- Extending [VkPipelineColorBlendStateCreateInfo](#):
 - [VkPipelineColorWriteCreateInfoEXT](#)

New Enum Constants

- [VK_EXT_COLOR_WRITE_ENABLE_EXTENSION_NAME](#)
- [VK_EXT_COLOR_WRITE_ENABLE_SPEC_VERSION](#)
- Extending [VkDynamicState](#):
 - [VK_DYNAMIC_STATE_COLOR_WRITE_ENABLE_EXT](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COLOR_WRITE_ENABLE_FEATURES_EXT](#)
 - [VK_STRUCTURE_TYPE_PIPELINE_COLOR_WRITE_CREATE_INFO_EXT](#)

Version History

- Revision 1, 2020-01-25 (Sharif Elcott)
 - Internal revisions

VK_EXT_conservative_rasterization

Name String

VK_EXT_conservative_rasterization

Extension Type

Device extension

Registered Extension Number

102

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

Vulkan Version 1.1

SPIR-V Dependencies

- SPV_EXT_fragment_fully_covered

Contact

- Piers Daniell [[GitHub](#)][pdaniell-nv](#)

Other Extension Metadata

Last Modified Date

2020-06-09

Interactions and External Dependencies

- This extension requires `SPV_EXT_fragment_fully_covered` if the `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::fullyCoveredFragmentShaderInputVariable` feature is used.
- This extension requires `SPV_KHR_post_depth_coverage` if the `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::conservativeRasterizationPostDepthCoverage` feature is used.
- This extension provides API support for `GL_NV_conservative_raster_underestimation` if the `VkPhysicalDeviceConservativeRasterizationPropertiesEXT::fullyCoveredFragmentShaderInputVariable` feature is used.

Contributors

- Daniel Koch, NVIDIA
- Daniel Rakos, AMD
- Jeff Bolz, NVIDIA
- Slawomir Grajewski, Intel
- Stu Smith, Imagination Technologies

Description

This extension adds a new rasterization mode called conservative rasterization. There are two modes of conservative rasterization; overestimation and underestimation.

When overestimation is enabled, if any part of the primitive, including its edges, covers any part of the rectangular pixel area, including its sides, then a fragment is generated with all coverage samples turned on. This extension allows for some variation in implementations by accounting for differences in overestimation, where the generating primitive size is increased at each of its edges by some sub-pixel amount to further increase conservative pixel coverage. Implementations can allow the application to specify an extra overestimation beyond the base overestimation the implementation already does. It also allows implementations to either cull degenerate primitives or rasterize them.

When underestimation is enabled, fragments are only generated if the rectangular pixel area is fully covered by the generating primitive. If supported by the implementation, when a pixel rectangle is fully covered the fragment shader input variable builtin called FullyCoveredEXT is set to true. The shader variable works in either overestimation or underestimation mode.

Implementations can process degenerate triangles and lines by either discarding them or generating conservative fragments for them. Degenerate triangles are those that end up with zero area after the rasterizer quantizes them to the fixed-point pixel grid. Degenerate lines are those with zero length after quantization.

New Structures

- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDeviceConservativeRasterizationPropertiesEXT](#)
- Extending [VkPipelineRasterizationStateCreateInfo](#):
 - [VkPipelineRasterizationConservativeStateCreateInfoEXT](#)

New Enums

- [VkConservativeRasterizationModeEXT](#)

New Bitmasks

- [VkPipelineRasterizationConservativeStateCreateFlagsEXT](#)

New Enum Constants

- `VK_EXT_CONSERVATIVE_RASTERIZATION_EXTENSION_NAME`
- `VK_EXT_CONSERVATIVE_RASTERIZATION_SPEC_VERSION`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CONSERVATIVE_RASTERIZATION_PROPERTIES_EXT`
 - `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_CONSERVATIVE_STATE_CREATE_INFO_EXT`

New Built-In Variables

- `FullyCoveredEXT`

New SPIR-V Capabilities

- `FragmentFullyCoveredEXT`

Version History

- Revision 1.1, 2020-09-06 (Piers Daniell)
 - Add missing SPIR-V and GLSL dependencies.
- Revision 1, 2017-08-28 (Piers Daniell)
 - Internal revisions

VK_EXT_custom_border_color

Name String

`VK_EXT_custom_border_color`

Extension Type

Device extension

Registered Extension Number

288

Revision

12

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_get_physical_device_properties2`

or

`Vulkan Version 1.1`

Special Uses

- `OpenGL / ES support`

- [D3D support](#)

Contact

- Liam Middlebrook [\[GitHub\]liam-middlebrook](#)

Other Extension Metadata

Last Modified Date

2020-04-16

IP Status

No known IP claims.

Contributors

- Joshua Ashton, Valve
- Hans-Kristian Arntzen, Valve
- Philip Rebohle, Valve
- Liam Middlebrook, NVIDIA
- Jeff Bolz, NVIDIA
- Tobias Hector, AMD
- Faith Ekstrand, Intel
- Spencer Fricke, Samsung Electronics
- Graeme Leese, Broadcom
- Jesse Hall, Google
- Jan-Harald Fredriksen, ARM
- Tom Olson, ARM
- Stuart Smith, Imagination Technologies
- Donald Scorgie, Imagination Technologies
- Alex Walters, Imagination Technologies
- Peter Quayle, Imagination Technologies

Description

This extension provides cross-vendor functionality to specify a custom border color for use when the sampler address mode `VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER` is used.

To create a sampler which uses a custom border color set `VkSamplerCreateInfo::borderColor` to one of:

- `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT`
- `VK_BORDER_COLOR_INT_CUSTOM_EXT`

When `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT` or `VK_BORDER_COLOR_INT_CUSTOM_EXT` is used, applications

must provide a `VkSamplerCustomBorderColorCreateInfoEXT` in the `pNext` chain for `VkSamplerCreateInfo`.

New Structures

- Extending `VkPhysicalDeviceFeatures2`, `VkDeviceCreateInfo`:
 - `VkPhysicalDeviceCustomBorderColorFeaturesEXT`
- Extending `VkPhysicalDeviceProperties2`:
 - `VkPhysicalDeviceCustomBorderColorPropertiesEXT`
- Extending `VkSamplerCreateInfo`:
 - `VkSamplerCustomBorderColorCreateInfoEXT`

New Enum Constants

- `VK_EXT_CUSTOM_BORDER_COLOR_EXTENSION_NAME`
- `VK_EXT_CUSTOM_BORDER_COLOR_SPEC_VERSION`
- Extending `VkBorderColor`:
 - `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT`
 - `VK_BORDER_COLOR_INT_CUSTOM_EXT`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUSTOM_BORDER_COLOR_FEATURES_EXT`
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_CUSTOM_BORDER_COLOR_PROPERTIES_EXT`
 - `VK_STRUCTURE_TYPE_SAMPLER_CUSTOM_BORDER_COLOR_CREATE_INFO_EXT`

Issues

1) Should `VkClearColorValue` be used for the border color value, or should we have our own struct/union? Do we need to specify the type of the input values for the components? This is more of a concern if `VkClearColorValue` is used here because it provides a union of float,int,uint types.

RESOLVED: Will reuse existing `VkClearColorValue` structure in order to easily take advantage of float,int,uint border color types.

2) For hardware which supports a limited number of border colors what happens if that number is exceeded? Should this be handled by the driver unbeknownst to the application? In Revision 1 we had solved this issue using a new Object type, however that may have lead to additional system resource consumption which would otherwise not be required.

RESOLVED: Added `VkPhysicalDeviceCustomBorderColorPropertiesEXT::maxCustomBorderColorSamplers` for tracking implementation-specific limit, and Valid Usage statement handling overflow.

3) Should this be supported for immutable samplers at all, or by a feature bit? Some implementations may not be able to support custom border colors on immutable samplers — is it worthwhile enabling this to work on them for implementations that can support it, or forbidding it entirely.

RESOLVED: Samplers created with a custom border color are forbidden from being immutable. This resolves concerns for implementations where the custom border color is an index to a LUT instead of being directly embedded into sampler state.

4) Should UINT and SINT (unsigned integer and signed integer) border color types be separated or should they be combined into one generic INT (integer) type?

RESOLVED: Separating these does not make much sense as the existing fixed border color types do not have this distinction, and there is no reason in hardware to do so. This separation would also create unnecessary work and considerations for the application.

Version History

- Revision 1, 2019-10-10 (Joshua Ashton)
 - Internal revisions.
- Revision 2, 2019-10-11 (Liam Middlebrook)
 - Remove `VkCustomBorderColor` object and associated functions
 - Add issues concerning HW limitations for custom border color count
- Revision 3, 2019-10-12 (Joshua Ashton)
 - Re-expose the limits for the maximum number of unique border colors
 - Add extra details about border color tracking
 - Fix typos
- Revision 4, 2019-10-12 (Joshua Ashton)
 - Changed `maxUniqueCustomBorderColors` to a `uint32_t` from a `VkDeviceSize`
- Revision 5, 2019-10-14 (Liam Middlebrook)
 - Added features bit
- Revision 6, 2019-10-15 (Joshua Ashton)
 - Type-ize `VK_BORDER_COLOR_CUSTOM`
 - Fix const-ness on `pNext` of `VkSamplerCustomBorderColorCreateInfoEXT`
- Revision 7, 2019-11-26 (Liam Middlebrook)
 - Renamed `maxUniqueCustomBorderColors` to `maxCustomBorderColors`
- Revision 8, 2019-11-29 (Joshua Ashton)
 - Renamed `borderColor` member of `VkSamplerCustomBorderColorCreateInfoEXT` to `customBorderColor`
- Revision 9, 2020-02-19 (Joshua Ashton)
 - Renamed `maxCustomBorderColors` to `maxCustomBorderColorSamplers`
- Revision 10, 2020-02-21 (Joshua Ashton)
 - Added format to `VkSamplerCustomBorderColorCreateInfoEXT` and feature bit
- Revision 11, 2020-04-07 (Joshua Ashton)

- Dropped UINT/SINT border color differences, consolidated types
- Revision 12, 2020-04-16 (Joshua Ashton)
 - Renamed `VK_BORDER_COLOR_CUSTOM_FLOAT_EXT` to `VK_BORDER_COLOR_FLOAT_CUSTOM_EXT` for consistency

VK_EXT_debug_utils

Name String

`VK_EXT_debug_utils`

Extension Type

Instance extension

Registered Extension Number

129

Revision

2

Ratification Status

Ratified

Extension and Version Dependencies

None

Special Use

- [Debugging tools](#)

Contact

- Mark Young [\[GitHub\]marky-lunarg](#)

Other Extension Metadata

Last Modified Date

2020-04-03

Revision

2

IP Status

No known IP claims.

Dependencies

- This extension is written against version 1.0 of the Vulkan API.
- Requires [VkObjectType](#)

Contributors

- Mark Young, LunarG

- Baldur Karlsson
- Ian Elliott, Google
- Courtney Goeltzenleuchter, Google
- Karl Schultz, LunarG
- Mark Lobodzinski, LunarG
- Mike Schuchardt, LunarG
- Jaakko Konttinen, AMD
- Dan Ginsburg, Valve Software
- Rolando Olivares, Epic Games
- Dan Baker, Oxide Games
- Kyle Spagnoli, NVIDIA
- Jon Ashburn, LunarG
- Piers Daniell, NVIDIA

Description

Due to the nature of the Vulkan interface, there is very little error information available to the developer and application. By using the `VK_EXT_debug_utils` extension, developers **can** obtain more information. When combined with validation layers, even more detailed feedback on the application's use of Vulkan will be provided.

This extension provides the following capabilities:

- The ability to create a debug messenger which will pass along debug messages to an application supplied callback.
- The ability to identify specific Vulkan objects using a name or tag to improve tracking.
- The ability to identify specific sections within a `VkQueue` or `VkCommandBuffer` using labels to aid organization and offline analysis in external tools.

The main difference between this extension and `VK_EXT_debug_report` and `VK_EXT_debug_marker` is that those extensions use `VkDebugReportObjectTypeEXT` to identify objects. This extension uses the core `VkObjectType` in place of `VkDebugReportObjectTypeEXT`. The primary reason for this move is that no future object type handle enumeration values will be added to `VkDebugReportObjectTypeEXT` since the creation of `VkObjectType`.

In addition, this extension combines the functionality of both `VK_EXT_debug_report` and `VK_EXT_debug_marker` by allowing object name and debug markers (now called labels) to be returned to the application's callback function. This should assist in clarifying the details of a debug message including: what objects are involved and potentially which location within a `VkQueue` or `VkCommandBuffer` the message occurred.

New Object Types

- `VkDebugUtilsMessengerEXT`

New Commands

- [vkCmdBeginDebugUtilsLabelEXT](#)
- [vkCmdEndDebugUtilsLabelEXT](#)
- [vkCmdInsertDebugUtilsLabelEXT](#)
- [vkCreateDebugUtilsMessengerEXT](#)
- [vkDestroyDebugUtilsMessengerEXT](#)
- [vkQueueBeginDebugUtilsLabelEXT](#)
- [vkQueueEndDebugUtilsLabelEXT](#)
- [vkQueueInsertDebugUtilsLabelEXT](#)
- [vkSetDebugUtilsObjectNameEXT](#)
- [vkSetDebugUtilsObjectTagEXT](#)
- [vkSubmitDebugUtilsMessageEXT](#)

New Structures

- [VkDebugUtilsLabelEXT](#)
- [VkDebugUtilsMessengerCallbackDataEXT](#)
- [VkDebugUtilsObjectTagInfoEXT](#)
- Extending [VkInstanceCreateInfo](#):
 - [VkDebugUtilsMessengerCreateInfoEXT](#)
- Extending [VkPipelineShaderStageCreateInfo](#):
 - [VkDebugUtilsObjectNameInfoEXT](#)

New Function Pointers

- [PFN_vkDebugUtilsMessengerCallbackEXT](#)

New Enums

- [VkDebugUtilsMessageSeverityFlagBitsEXT](#)
- [VkDebugUtilsMessageTypeFlagBitsEXT](#)

New Bitmasks

- [VkDebugUtilsMessageSeverityFlagsEXT](#)
- [VkDebugUtilsMessageTypeFlagsEXT](#)
- [VkDebugUtilsMessengerCallbackDataFlagsEXT](#)
- [VkDebugUtilsMessengerCreateFlagsEXT](#)

New Enum Constants

- `VK_EXT_DEBUG_UTILS_EXTENSION_NAME`
- `VK_EXT_DEBUG_UTILS_SPEC_VERSION`
- Extending `VkObjectType`:
 - `VK_OBJECT_TYPE_DEBUG_UTILS_MESSENGER_EXT`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT`
 - `VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CALLBACK_DATA_EXT`
 - `VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT`
 - `VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT`
 - `VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_TAG_INFO_EXT`

Examples

Example 1

`VK_EXT_debug_utils` allows an application to register multiple callbacks with any Vulkan component wishing to report debug information. Some callbacks may log the information to a file, others may cause a debug break point or other application-defined behavior. An application **can** register callbacks even when no validation layers are enabled, but they will only be called for loader and, if implemented, driver events.

To capture events that occur while creating or destroying an instance an application **can** link a `VkDebugUtilsMessengerCreateInfoEXT` structure to the `pNext` chain of the `VkInstanceCreateInfo` structure passed to `vkCreateInstance`.

Example uses: Create three callback objects. One will log errors and warnings to the debug console using Windows `OutputDebugString`. The second will cause the debugger to break at that callback when an error happens and the third will log warnings to stdout.

```
extern VkInstance instance;
VkResult res;
VkDebugUtilsMessengerEXT cb1, cb2, cb3;

// Must call extension functions through a function pointer:
PFN_vkCreateDebugUtilsMessengerEXT pfnCreateDebugUtilsMessengerEXT =
(PFN_vkCreateDebugUtilsMessengerEXT)vkGetInstanceProcAddr(instance,
"vkCreateDebugUtilsMessengerEXT");
PFN_vkDestroyDebugUtilsMessengerEXT pfnDestroyDebugUtilsMessengerEXT =
(PFN_vkDestroyDebugUtilsMessengerEXT)vkGetInstanceProcAddr(instance,
"vkDestroyDebugUtilsMessengerEXT");

VkDebugUtilsMessengerCreateInfoEXT callback1 = {
    .sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT,
    .pNext = NULL,
```

```

        .flags = 0,
        .messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT |
                           VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT,
        .messageType = VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT |
                       VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT,
        .pfnUserCallback = myOutputDebugString,
        .pUserData = NULL
    };
    res = pfncreateDebugUtilsMessengerEXT(instance, &callback1, NULL, &cb1);
    if (res != VK_SUCCESS) {
        // Do error handling for VK_ERROR_OUT_OF_MEMORY
    }

    callback1.messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT;
    callback1.pfnUserCallback = myDebugBreak;
    callback1.pUserData = NULL;
    res = pfncreateDebugUtilsMessengerEXT(instance, &callback1, NULL, &cb2);
    if (res != VK_SUCCESS) {
        // Do error handling for VK_ERROR_OUT_OF_MEMORY
    }

    VkDebugUtilsMessengerCreateInfoEXT callback3 = {
        .sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT,
        .pNext = NULL,
        .flags = 0,
        .messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT,
        .messageType = VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT |
                      VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT,
        .pfnUserCallback = myStdOutLogger,
        .pUserData = NULL
    };
    res = pfncreateDebugUtilsMessengerEXT(instance, &callback3, NULL, &cb3);
    if (res != VK_SUCCESS) {
        // Do error handling for VK_ERROR_OUT_OF_MEMORY
    }

    ...

    // Remove callbacks when cleaning up
    pfndestroyDebugUtilsMessengerEXT(instance, cb1, NULL);
    pfndestroyDebugUtilsMessengerEXT(instance, cb2, NULL);
    pfndestroyDebugUtilsMessengerEXT(instance, cb3, NULL);

```

Example 2

Associate a name with an image, for easier debugging in external tools or with validation layers that can print a friendly name when referring to objects in error messages.

```

extern VkInstance instance;
extern VkDevice device;

```

```

extern VkImage image;

// Must call extension functions through a function pointer:
PFN_vkSetDebugUtilsObjectNameEXT pfnSetDebugUtilsObjectNameEXT =
(PFN_vkSetDebugUtilsObjectNameEXT)vkGetInstanceProcAddr(instance,
"vkSetDebugUtilsObjectNameEXT");

// Set a name on the image
const VkDebugUtilsObjectNameInfoEXT imageNameInfo =
{
    .sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_OBJECT_NAME_INFO_EXT,
    .pNext = NULL,
    .objectType = VK_OBJECT_TYPE_IMAGE,
    .objectHandle = (uint64_t)image,
    .pObjectName = "Brick Diffuse Texture",
};

pfnSetDebugUtilsObjectNameEXT(device, &imageNameInfo);

// A subsequent error might print:
// Image 'Brick Diffuse Texture' (0xc0dec0dedeadbeef) is used in a
// command buffer with no memory bound to it.

```

Example 3

Annotating regions of a workload with naming information so that offline analysis tools can display a more usable visualization of the commands submitted.

```

extern VkInstance instance;
extern VkCommandBuffer commandBuffer;

// Must call extension functions through a function pointer:
PFN_vkQueueBeginDebugUtilsLabelEXT pfnQueueBeginDebugUtilsLabelEXT =
(PFN_vkQueueBeginDebugUtilsLabelEXT)vkGetInstanceProcAddr(instance,
"vkQueueBeginDebugUtilsLabelEXT");
PFN_vkQueueEndDebugUtilsLabelEXT pfnQueueEndDebugUtilsLabelEXT =
(PFN_vkQueueEndDebugUtilsLabelEXT)vkGetInstanceProcAddr(instance,
"vkQueueEndDebugUtilsLabelEXT");
PFN_vkCmdBeginDebugUtilsLabelEXT pfnCmdBeginDebugUtilsLabelEXT =
(PFN_vkCmdBeginDebugUtilsLabelEXT)vkGetInstanceProcAddr(instance,
"vkCmdBeginDebugUtilsLabelEXT");
PFN_vkCmdEndDebugUtilsLabelEXT pfnCmdEndDebugUtilsLabelEXT =
(PFN_vkCmdEndDebugUtilsLabelEXT)vkGetInstanceProcAddr(instance,
"vkCmdEndDebugUtilsLabelEXT");
PFN_vkCmdInsertDebugUtilsLabelEXT pfnCmdInsertDebugUtilsLabelEXT =
(PFN_vkCmdInsertDebugUtilsLabelEXT)vkGetInstanceProcAddr(instance,
"vkCmdInsertDebugUtilsLabelEXT");

// Describe the area being rendered
const VkDebugUtilsLabelEXT houseLabel =

```

```

{
    .sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT,
    .pNext = NULL,
    .pLabelName = "Brick House",
    .color = { 1.0f, 0.0f, 0.0f, 1.0f },
};

// Start an annotated group of calls under the 'Brick House' name
pfnCmdBeginDebugUtilsLabelEXT(commandBuffer, &houseLabel);
{
    // A mutable structure for each part being rendered
    VkDebugUtilsLabelEXT housePartLabel =
    {
        .sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT,
        .pNext = NULL,
        .pLabelName = NULL,
        .color = { 0.0f, 0.0f, 0.0f, 0.0f },
    };

    // Set the name and insert the marker
    housePartLabel.pLabelName = "Walls";
    pfnCmdInsertDebugUtilsLabelEXT(commandBuffer, &housePartLabel);

    // Insert the drawcall for the walls
    vkCmdDrawIndexed(commandBuffer, 1000, 1, 0, 0, 0);

    // Insert a recursive region for two sets of windows
    housePartLabel.pLabelName = "Windows";
    pfnCmdBeginDebugUtilsLabelEXT(commandBuffer, &housePartLabel);
    {
        vkCmdDrawIndexed(commandBuffer, 75, 6, 1000, 0, 0);
        vkCmdDrawIndexed(commandBuffer, 100, 2, 1450, 0, 0);
    }
    pfnCmdEndDebugUtilsLabelEXT(commandBuffer);

    housePartLabel.pLabelName = "Front Door";
    pfnCmdInsertDebugUtilsLabelEXT(commandBuffer, &housePartLabel);

    vkCmdDrawIndexed(commandBuffer, 350, 1, 1650, 0, 0);

    housePartLabel.pLabelName = "Roof";
    pfnCmdInsertDebugUtilsLabelEXT(commandBuffer, &housePartLabel);

    vkCmdDrawIndexed(commandBuffer, 500, 1, 2000, 0, 0);
}
// End the house annotation started above
pfnCmdEndDebugUtilsLabelEXT(commandBuffer);

// Do other work

vkEndCommandBuffer(commandBuffer);

```

```

// Describe the queue being used
const VkDebugUtilsLabelEXT queueLabel =
{
    .sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_LABEL_EXT,
    .pNext = NULL,
    .pLabelName = "Main Render Work",
    .color = { 0.0f, 1.0f, 0.0f, 1.0f },
};

// Identify the queue label region
pfnQueueBeginDebugUtilsLabelEXT(queue, &queueLabel);

// Submit the work for the main render thread
const VkCommandBuffer cmd_bufs[] = {commandBuffer};
VkSubmitInfo submit_info =
{
    .sType = VK_STRUCTURE_TYPE_SUBMIT_INFO,
    .pNext = NULL,
    .waitSemaphoreCount = 0,
    .pWaitSemaphores = NULL,
    .pWaitDstStageMask = NULL,
    .commandBufferCount = 1,
    .pCommandBuffers = cmd_bufs,
    .signalSemaphoreCount = 0,
    .pSignalSemaphores = NULL
};
vkQueueSubmit(queue, 1, &submit_info, fence);

// End the queue label region
pfnQueueEndDebugUtilsLabelEXT(queue);

```

Issues

1) Should we just name this extension `VK_EXT_debug_report2`

RESOLVED: No. There is enough additional changes to the structures to break backwards compatibility. So, a new name was decided that would not indicate any interaction with the previous extension.

2) Will validation layers immediately support all the new features.

RESOLVED: Not immediately. As one can imagine, there is a lot of work involved with converting the validation layer logging over to the new functionality. Basic logging, as seen in the origin `VK_EXT_debug_report` extension will be made available immediately. However, adding the labels and object names will take time. Since the priority for Khronos at this time is to continue focusing on Valid Usage statements, it may take a while before the new functionality is fully exposed.

3) If the validation layers will not expose the new functionality immediately, then what is the point of this extension?

RESOLVED: We needed a replacement for `VK_EXT_debug_report` because the `VkDebugReportObjectTypeEXT` enumeration will no longer be updated and any new objects will need to be debugged using the new functionality provided by this extension.

4) Should this extension be split into two separate parts (1 extension that is an instance extension providing the callback functionality, and another device extension providing the general debug marker and annotation functionality)?

RESOLVED: No, the functionality for this extension is too closely related. If we did split up the extension, where would the structures and enums live, and how would you define that the device behavior in the instance extension is really only valid if the device extension is enabled, and the functionality is passed in. It is cleaner to just define this all as an instance extension, plus it allows the application to enable all debug functionality provided with one enable string during [vkCreateInstance](#).

Version History

- Revision 1, 2017-09-14 (Mark Young and all listed Contributors)
 - Initial draft, based on `VK_EXT_debug_report` and `VK_EXT_debug_marker` in addition to previous feedback supplied from various companies including Valve, Epic, and Oxide games.
- Revision 2, 2020-04-03 (Mark Young and Piers Daniell)
 - Updated to allow either `NULL` or an empty string to be passed in for `pObjectName` in `VkDebugUtilsObjectNameInfoEXT`, because the loader and various drivers support `NULL` already.

VK_EXT_depth_clip_enable

Name String

`VK_EXT_depth_clip_enable`

Extension Type

Device extension

Registered Extension Number

103

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_get_physical_device_properties2`

or

[Vulkan Version 1.1](#)

Special Use

- [D3D support](#)

Contact

- Piers Daniell [\[GitHub\]](#)[pdaniell-nv](#)

Other Extension Metadata

Last Modified Date

2018-12-20

Contributors

- Daniel Rakos, AMD
- Henri Verbeet, CodeWeavers
- Jeff Bolz, NVIDIA
- Philip Rebohle, DXVK
- Tobias Hector, AMD

Description

This extension allows the depth clipping operation, that is normally implicitly controlled by [VkPipelineRasterizationStateCreateInfo::depthClampEnable](#), to instead be controlled explicitly by [VkPipelineRasterizationDepthClipStateCreateInfoEXT::depthClipEnable](#).

This is useful for translating DX content which assumes depth clamping is always enabled, but depth clip can be controlled by the DepthClipEnable rasterization state (D3D12_RASTERIZER_DESC).

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceDepthClipEnableFeaturesEXT](#)
- Extending [VkPipelineRasterizationStateCreateInfo](#):
 - [VkPipelineRasterizationDepthClipStateCreateInfoEXT](#)

New Bitmasks

- [VkPipelineRasterizationDepthClipStateCreateFlagsEXT](#)

New Enum Constants

- [VK_EXT_DEPTH_CLIP_ENABLE_EXTENSION_NAME](#)
- [VK_EXT_DEPTH_CLIP_ENABLE_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DEPTH_CLIP_ENABLE_FEATURES_EXT](#)
 - [VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_DEPTH_CLIP_STATE_CREATE_INFO_EXT](#)

Version History

- Revision 1, 2018-12-20 (Piers Daniell)
 - Internal revisions

VK_EXT_depth_range_unrestricted

Name String

VK_EXT_depth_range_unrestricted

Extension Type

Device extension

Registered Extension Number

14

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

None

Contact

- Piers Daniell [[GitHub](#)]pdaniell-nv

Other Extension Metadata

Last Modified Date

2017-06-22

Contributors

- Daniel Koch, NVIDIA
- Jeff Bolz, NVIDIA

Description

This extension removes the [VkViewport](#) `minDepth` and `maxDepth` restrictions that the values must be between `0.0` and `1.0`, inclusive. It also removes the same restriction on [VkPipelineDepthStencilStateCreateInfo](#) `minDepthBounds` and `maxDepthBounds`. Finally it removes the restriction on the `depth` value in [VkClearDepthStencilValue](#).

New Enum Constants

- VK_EXT_DEPTH_RANGE_UNRESTRICTED_EXTENSION_NAME
- VK_EXT_DEPTH_RANGE_UNRESTRICTED_SPEC_VERSION

Issues

1) How do [VkViewport minDepth](#) and [maxDepth](#) values outside of the [0.0](#) to [1.0](#) range interact with [Primitive Clipping](#)?

RESOLVED: The behavior described in [Primitive Clipping](#) still applies. If depth clamping is disabled the depth values are still clipped to $0 \leq z_c \leq w_c$ before the viewport transform. If depth clamping is enabled the above equation is ignored and the depth values are instead clamped to the [VkViewport minDepth](#) and [maxDepth](#) values, which in the case of this extension can be outside of the [0.0](#) to [1.0](#) range.

2) What happens if a resulting depth fragment is outside of the [0.0](#) to [1.0](#) range and the depth buffer is fixed-point rather than floating-point?

RESOLVED: This situation can also arise without this extension (when fragment shaders replace depth values, for example), and this extension does not change the behavior, which is defined in the [Depth Test](#) section of the Fragment Operations chapter.

Version History

- Revision 1, 2017-06-22 (Piers Daniell)
 - Internal revisions

VK_EXT_direct_mode_display

Name String

[VK_EXT_direct_mode_display](#)

Extension Type

Instance extension

Registered Extension Number

89

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_display](#)

Contact

- James Jones [\[GitHub\]](#)[cubanismo](#)

Other Extension Metadata

Last Modified Date

2016-12-13

IP Status

No known IP claims.

Contributors

- Pierre Boudier, NVIDIA
- James Jones, NVIDIA
- Damien Leone, NVIDIA
- Pierre-Loup Griffais, Valve
- Liam Middlebrook, NVIDIA

Description

This is extension, along with related platform extensions, allows applications to take exclusive control of displays associated with a native windowing system. This is especially useful for virtual reality applications that wish to hide HMDs (head mounted displays) from the native platform's display management system, desktop, and/or other applications.

New Commands

- [vkReleaseDisplayEXT](#)

New Enum Constants

- [VK_EXT_DIRECT_MODE_DISPLAY_EXTENSION_NAME](#)
- [VK_EXT_DIRECT_MODE_DISPLAY_SPEC_VERSION](#)

Issues

1) Should this extension and its related platform-specific extensions leverage [VK_KHR_display](#), or provide separate equivalent interfaces.

RESOLVED: Use [VK_KHR_display](#) concepts and objects. [VK_KHR_display](#) can be used to enumerate all displays on the system, including those attached to/in use by a window system or native platform, but [VK_KHR_display_swapchain](#) will fail to create a swapchain on in-use displays. This extension and its platform-specific children will allow applications to grab in-use displays away from window systems and/or native platforms, allowing them to be used with [VK_KHR_display_swapchain](#).

2) Are separate calls needed to acquire displays and enable direct mode?

RESOLVED: No, these operations happen in one combined command. Acquiring a display puts it into direct mode.

Version History

- Revision 1, 2016-12-13 (James Jones)

- Initial draft

VK_EXT_discard_rectangles

Name String

VK_EXT_discard_rectangles

Extension Type

Device extension

Registered Extension Number

100

Revision

2

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

[Vulkan Version 1.1](#)

Contact

- Piers Daniell [\[GitHub\]](#)pdaniell-nv

Other Extension Metadata

Last Modified Date

2023-01-18

Interactions and External Dependencies

- Interacts with [VK_KHR_device_group](#)
- Interacts with Vulkan 1.1

Contributors

- Daniel Koch, NVIDIA
- Jeff Bolz, NVIDIA

Description

This extension provides additional orthogonally aligned “discard rectangles” specified in framebuffer-space coordinates that restrict rasterization of all points, lines and triangles.

From zero to an implementation-dependent limit (specified by [maxDiscardRectangles](#)) number of discard rectangles can be operational at once. When one or more discard rectangles are active, rasterized fragments can either survive if the fragment is within any of the operational discard

rectangles (`VK_DISCARD_RECTANGLE_MODE_INCLUSIVE_EXT` mode) or be rejected if the fragment is within any of the operational discard rectangles (`VK_DISCARD_RECTANGLE_MODE_EXCLUSIVE_EXT` mode).

These discard rectangles operate orthogonally to the existing scissor test functionality. The discard rectangles can be different for each physical device in a device group by specifying the device mask and setting discard rectangle dynamic state.

Version 2 of this extension introduces new dynamic states `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT` and `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT`, and the corresponding functions `vkCmdSetDiscardRectangleEnableEXT` and `vkCmdSetDiscardRectangleModeEXT`. Applications that use these dynamic states must ensure the implementation advertises at least `specVersion 2` of this extension.

New Commands

- `vkCmdSetDiscardRectangleEXT`
- `vkCmdSetDiscardRectangleEnableEXT`
- `vkCmdSetDiscardRectangleModeEXT`

New Structures

- Extending `VkGraphicsPipelineCreateInfo`:
 - `VkPipelineDiscardRectangleStateCreateInfoEXT`
- Extending `VkPhysicalDeviceProperties2`:
 - `VkPhysicalDeviceDiscardRectanglePropertiesEXT`

New Enums

- `VkDiscardRectangleModeEXT`

New Bitmasks

- `VkPipelineDiscardRectangleStateCreateFlagsEXT`

New Enum Constants

- `VK_EXT_DISCARD_RECTANGLES_EXTENSION_NAME`
- `VK_EXT_DISCARD_RECTANGLES_SPEC_VERSION`
- Extending `VkDynamicState`:
 - `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_ENABLE_EXT`
 - `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_EXT`
 - `VK_DYNAMIC_STATE_DISCARD_RECTANGLE_MODE_EXT`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_DISCARD_RECTANGLE_PROPERTIES_EXT`
 - `VK_STRUCTURE_TYPE_PIPELINE_DISCARD_RECTANGLE_STATE_CREATE_INFO_EXT`

Version History

- Revision 2, 2023-01-18 (Piers Daniell)
 - Add dynamic states for discard rectangle enable/disable and mode.
- Revision 1, 2016-12-22 (Piers Daniell)
 - Internal revisions

VK_EXT_display_control

Name String

VK_EXT_display_control

Extension Type

Device extension

Registered Extension Number

92

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_EXT_display_surface_counter
and
VK_KHR_swapchain

Contact

- James Jones [\[GitHub\]](#)cubanismo

Other Extension Metadata

Last Modified Date

2016-12-13

IP Status

No known IP claims.

Contributors

- Pierre Boudier, NVIDIA
- James Jones, NVIDIA
- Damien Leone, NVIDIA
- Pierre-Loup Griffais, Valve
- Daniel Vetter, Intel

Description

This extension defines a set of utility functions for use with the [VK_KHR_display](#) and [VK_KHR_display_swapchain](#) extensions.

New Commands

- [vkDisplayPowerControlEXT](#)
- [vkGetSwapchainCounterEXT](#)
- [vkRegisterDeviceEventEXT](#)
- [vkRegisterDisplayEventEXT](#)

New Structures

- [VkDeviceEventInfoEXT](#)
- [VkDisplayEventInfoEXT](#)
- [VkDisplayPowerInfoEXT](#)
- Extending [VkSwapchainCreateInfoKHR](#):
 - [VkSwapchainCounterCreateInfoEXT](#)

New Enums

- [VkDeviceEventTypeEXT](#)
- [VkDisplayEventTypeEXT](#)
- [VkDisplayPowerStateEXT](#)

New Enum Constants

- [VK_EXT_DISPLAY_CONTROL_EXTENSION_NAME](#)
- [VK_EXT_DISPLAY_CONTROL_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_DEVICE_EVENT_INFO_EXT](#)
 - [VK_STRUCTURE_TYPE_DISPLAY_EVENT_INFO_EXT](#)
 - [VK_STRUCTURE_TYPE_DISPLAY_POWER_INFO_EXT](#)
 - [VK_STRUCTURE_TYPE_SWAPCHAIN_COUNTER_CREATE_INFO_EXT](#)

Issues

1) Should this extension add an explicit “WaitForVsync” API or a fence signaled at vsync that the application can wait on?

RESOLVED: A fence. A separate API could later be provided that allows exporting the fence to a native object that could be inserted into standard run loops on POSIX and Windows systems.

2) Should callbacks be added for a vsync event, or in general to monitor events in Vulkan?

RESOLVED: No, fences should be used. Some events are generated by interrupts which are managed in the kernel. In order to use a callback provided by the application, drivers would need to have the userspace driver spawn threads that would wait on the kernel event, and hence the callbacks could be difficult for the application to synchronize with its other work given they would arrive on a foreign thread.

3) Should vblank or scanline events be exposed?

RESOLVED: Vblank events. Scanline events could be added by a separate extension, but the latency of processing an interrupt and waking up a userspace event is high enough that the accuracy of a scanline event would be rather low. Further, per-scanline interrupts are not supported by all hardware.

Version History

- Revision 1, 2016-12-13 (James Jones)
 - Initial draft

VK_EXT_display_surface_counter

Name String

VK_EXT_display_surface_counter

Extension Type

Instance extension

Registered Extension Number

91

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_display](#)

Contact

- James Jones [\[GitHub\]cubanismo](#)

Other Extension Metadata

Last Modified Date

2016-12-13

IP Status

No known IP claims.

Contributors

- Pierre Boudier, NVIDIA
- James Jones, NVIDIA
- Damien Leone, NVIDIA
- Pierre-Loup Griffais, Valve
- Daniel Vetter, Intel

Description

This extension defines a vertical blanking period counter associated with display surfaces. It provides a mechanism to query support for such a counter from a [VkSurfaceKHR](#) object.

New Commands

- [vkGetPhysicalDeviceSurfaceCapabilities2EXT](#)

New Structures

- [VkSurfaceCapabilities2EXT](#)

New Enums

- [VkSurfaceCounterFlagBitsEXT](#)

New Bitmasks

- [VkSurfaceCounterFlagsEXT](#)

New Enum Constants

- [VK_EXT_DISPLAY_SURFACE_COUNTER_EXTENSION_NAME](#)
- [VK_EXT_DISPLAY_SURFACE_COUNTER_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES_2_EXT](#)

Version History

- Revision 1, 2016-12-13 (James Jones)
 - Initial draft

VK_EXT_extended_dynamic_state

Name String

[VK_EXT_extended_dynamic_state](#)

Extension Type

Device extension

Registered Extension Number

268

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_get_physical_device_properties2](#)

or

[Vulkan Version 1.1](#)

Deprecation State

- *Promoted* to Vulkan 1.3

Contact

- Piers Daniell [\[GitHub\]](#)[pdaniell-nv](#)

Other Extension Metadata**Last Modified Date**

2019-12-09

IP Status

No known IP claims.

Contributors

- Dan Ginsburg, Valve Corporation
- Graeme Leese, Broadcom
- Hans-Kristian Arntzen, Valve Corporation
- Jan-Harald Fredriksen, Arm Limited
- Faith Ekstrand, Intel
- Jeff Bolz, NVIDIA
- Jesse Hall, Google
- Philip Rebohle, Valve Corporation
- Stuart Smith, Imagination Technologies
- Tobias Hector, AMD

Description

This extension adds some more dynamic state to support applications that need to reduce the number of pipeline state objects they compile and bind.

New Commands

- [vkCmdBindVertexBuffers2EXT](#)
- [vkCmdSetCullModeEXT](#)
- [vkCmdSetDepthBoundsTestEnableEXT](#)
- [vkCmdSetDepthCompareOpEXT](#)
- [vkCmdSetDepthTestEnableEXT](#)
- [vkCmdSetDepthWriteEnableEXT](#)
- [vkCmdSetFrontFaceEXT](#)
- [vkCmdSetPrimitiveTopologyEXT](#)
- [vkCmdSetScissorWithCountEXT](#)
- [vkCmdSetStencilOpEXT](#)
- [vkCmdSetStencilTestEnableEXT](#)
- [vkCmdSetViewportWithCountEXT](#)

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceExtendedDynamicStateFeaturesEXT](#)

New Enum Constants

- [VK_EXT_EXTENDED_DYNAMIC_STATE_EXTENSION_NAME](#)
- [VK_EXT_EXTENDED_DYNAMIC_STATE_SPEC_VERSION](#)
- Extending [VkDynamicState](#):
 - [VK_DYNAMIC_STATE_CULL_MODE_EXT](#)
 - [VK_DYNAMIC_STATE_DEPTH_BOUNDS_TEST_ENABLE_EXT](#)
 - [VK_DYNAMIC_STATE_DEPTH_COMPARE_OP_EXT](#)
 - [VK_DYNAMIC_STATE_DEPTH_TEST_ENABLE_EXT](#)
 - [VK_DYNAMIC_STATE_DEPTH_WRITE_ENABLE_EXT](#)
 - [VK_DYNAMIC_STATE_FRONT_FACE_EXT](#)
 - [VK_DYNAMIC_STATE_PRIMITIVE_TOPOLOGY_EXT](#)
 - [VK_DYNAMIC_STATE_SCISSOR_WITH_COUNT_EXT](#)
 - [VK_DYNAMIC_STATE_STENCIL_OP_EXT](#)
 - [VK_DYNAMIC_STATE_STENCIL_TEST_ENABLE_EXT](#)

- `VK_DYNAMIC_STATE_VERTEX_INPUT_BINDING_STRIDE_EXT`
- `VK_DYNAMIC_STATE_VIEWPORT_WITH_COUNT_EXT`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_FEATURES_EXT`

Promotion to Vulkan 1.3

All dynamic state enumerants and commands in this extension are included in core Vulkan 1.3, with the EXT suffix omitted. The feature structure is not promoted. Extension interfaces that were promoted remain available as aliases of the core functionality.

Issues

1) Why are the values of `pStrides` in `vkCmdBindVertexBuffers2EXT` limited to be between 0 and the maximum extent of the binding, when this restriction is not present for the same static state?

Implementing these edge cases adds overhead to some implementations that would require significant cost when calling this function, and the intention is that this state should be more or less free to change.

`VK_EXT_vertex_input_dynamic_state` allows the stride to be changed freely when supported via `vkCmdSetVertexInputEXT`.

Version History

- Revision 1, 2019-12-09 (Piers Daniell)
 - Internal revisions

VK_EXT_extended_dynamic_state2

Name String

`VK_EXT_extended_dynamic_state2`

Extension Type

Device extension

Registered Extension Number

378

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_get_physical_device_properties2`
or

Deprecation State

- *Promoted* to Vulkan 1.3

Contact

- Vikram Kushwaha [[GitHub](#)][vkushwaha-nv](#)

Other Extension Metadata

Last Modified Date

2021-04-12

IP Status

No known IP claims.

Contributors

- Vikram Kushwaha, NVIDIA
- Piers Daniell, NVIDIA
- Jeff Bolz, NVIDIA

Description

This extension adds some more dynamic state to support applications that need to reduce the number of pipeline state objects they compile and bind.

New Commands

- [vkCmdSetDepthBiasEnableEXT](#)
- [vkCmdSetLogicOpEXT](#)
- [vkCmdSetPatchControlPointsEXT](#)
- [vkCmdSetPrimitiveRestartEnableEXT](#)
- [vkCmdSetRasterizerDiscardEnableEXT](#)

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceExtendedDynamicState2FeaturesEXT](#)

New Enum Constants

- [VK_EXT_EXTENDED_DYNAMIC_STATE_2_EXTENSION_NAME](#)
- [VK_EXT_EXTENDED_DYNAMIC_STATE_2_SPEC_VERSION](#)
- Extending [VkDynamicState](#):
 - [VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE_EXT](#)

- `VK_DYNAMIC_STATE_LOGIC_OP_EXT`
- `VK_DYNAMIC_STATE_PATCH_CONTROL_POINTS_EXT`
- `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE_EXT`
- `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE_EXT`
- Extending [VkStructureType](#):
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTENDED_DYNAMIC_STATE_2_FEATURES_EXT`

Promotion to Vulkan 1.3

The dynamic state enumerants `VK_DYNAMIC_STATE_DEPTH_BIAS_ENABLE_EXT`, `VK_DYNAMIC_STATE_PRIMITIVE_RESTART_ENABLE_EXT`, and `VK_DYNAMIC_STATE_RASTERIZER_DISCARD_ENABLE_EXT`; and the corresponding commands in this extension are included in core Vulkan 1.3, with the EXT suffix omitted. The enumerants and commands for dynamic logic operation and patch control points are not promoted, nor is the feature structure. Extension interfaces that were promoted remain available as aliases of the core functionality.

Version History

- Revision 1, 2021-04-12 (Vikram Kushwaha)
 - Internal revisions

VK_EXT_external_memory_dma_buf

Name String

`VK_EXT_external_memory_dma_buf`

Extension Type

Device extension

Registered Extension Number

126

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_external_memory_fd](#)

Contact

- Lina Versace [[GitHub](#)][linyaa-kiwi](#)

Other Extension Metadata

Last Modified Date

2017-10-10

IP Status

No known IP claims.

Contributors

- Lina Versace, Google
- James Jones, NVIDIA
- Faith Ekstrand, Intel

Description

A `dma_buf` is a type of file descriptor, defined by the Linux kernel, that allows sharing memory across kernel device drivers and across processes. This extension enables applications to import a `dma_buf` as `VkDeviceMemory`, to export `VkDeviceMemory` as a `dma_buf`, and to create `VkBuffer` objects that **can** be bound to that memory.

New Enum Constants

- `VK_EXT_EXTERNAL_MEMORY_DMA_BUF_EXTENSION_NAME`
- `VK_EXT_EXTERNAL_MEMORY_DMA_BUF_SPEC_VERSION`
- Extending `VkExternalMemoryHandleTypeFlagBits`:
 - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_DMA_BUF_BIT_EXT`

Issues

1) How does the application, when creating a `VkImage` that it intends to bind to `dma_buf` `VkDeviceMemory` containing an externally produced image, specify the memory layout (such as row pitch and DRM format modifier) of the `VkImage`? In other words, how does the application achieve behavior comparable to that provided by `EGL_EXT_image_dma_buf_import` and `EGL_EXT_image_dma_buf_import_modifiers`?

RESOLVED: Features comparable to those in `EGL_EXT_image_dma_buf_import` and `EGL_EXT_image_dma_buf_import_modifiers` will be provided by an extension layered atop this one.

2) Without the ability to specify the memory layout of external `dma_buf` images, how is this extension useful?

RESOLVED: This extension provides exactly one new feature: the ability to import/export between `dma_buf` and `VkDeviceMemory`. This feature, together with features provided by `VK_KHR_external_memory_fd`, is sufficient to bind a `VkBuffer` to `dma_buf`.

Version History

- Revision 1, 2017-10-10 (Lina Versace)

- Squashed internal revisions

VK_EXT_external_memory_host

Name String

VK_EXT_external_memory_host

Extension Type

Device extension

Registered Extension Number

179

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_external_memory

or

[Vulkan Version 1.1](#)

Contact

- Daniel Rakos [\[GitHub\]drakos-amd](#)

Other Extension Metadata

Last Modified Date

2017-11-10

IP Status

No known IP claims.

Contributors

- Jaakko Konttinen, AMD
- David Mao, AMD
- Daniel Rakos, AMD
- Tobias Hector, Imagination Technologies
- Faith Ekstrand, Intel
- James Jones, NVIDIA

Description

This extension enables an application to import host allocations and host mapped foreign device

memory to Vulkan memory objects.

New Commands

- [vkGetMemoryHostPointerPropertiesEXT](#)

New Structures

- [VkMemoryHostPointerPropertiesEXT](#)
- Extending [VkMemoryAllocateInfo](#):
 - [VkImportMemoryHostPointerInfoEXT](#)
- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDeviceExternalMemoryHostPropertiesEXT](#)

New Enum Constants

- [VK_EXT_EXTERNAL_MEMORY_HOST_EXTENSION_NAME](#)
- [VK_EXT_EXTERNAL_MEMORY_HOST_SPEC_VERSION](#)
- Extending [VkExternalMemoryHandleTypeFlagBits](#):
 - [VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_ALLOCATION_BIT_EXT](#)
 - [VK_EXTERNAL_MEMORY_HANDLE_TYPE_HOST_MAPPED_FOREIGN_MEMORY_BIT_EXT](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_IMPORT_MEMORY_HOST_POINTER_INFO_EXT](#)
 - [VK_STRUCTURE_TYPE_MEMORY_HOST_POINTER_PROPERTIES_EXT](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_HOST_PROPERTIES_EXT](#)

Issues

1) What memory type has to be used to import host pointers?

RESOLVED: Depends on the implementation. Applications have to use the new [vkGetMemoryHostPointerPropertiesEXT](#) command to query the supported memory types for a particular host pointer. The reported memory types may include memory types that come from a memory heap that is otherwise not usable for regular memory object allocation and thus such a heap's size may be zero.

2) Can the application still access the contents of the host allocation after importing?

RESOLVED: Yes. However, usual synchronization requirements apply.

3) Can the application free the host allocation?

RESOLVED: No, it violates valid usage conditions. Using the memory object imported from a host allocation that is already freed thus results in undefined behavior.

4) Is [vkMapMemory](#) expected to return the same host address which was specified when importing

it to the memory object?

RESOLVED: No. Implementations are allowed to return the same address but it is not required. Some implementations might return a different virtual mapping of the allocation, although the same physical pages will be used.

5) Is there any limitation on the alignment of the host pointer and/or size?

RESOLVED: Yes. Both the address and the size have to be an integer multiple of `minImportedHostPointerAlignment`. In addition, some platforms and foreign devices may have additional restrictions.

6) Can the same host allocation be imported multiple times into a given physical device?

RESOLVED: No, at least not guaranteed by this extension. Some platforms do not allow locking the same physical pages for device access multiple times, so attempting to do it may result in undefined behavior.

7) Does this extension support exporting the new handle type?

RESOLVED: No.

8) Should we include the possibility to import host mapped foreign device memory using this API?

RESOLVED: Yes, through a separate handle type. Implementations are still allowed to support only one of the handle types introduced by this extension by not returning import support for a particular handle type as returned in `VkExternalMemoryPropertiesKHR`.

Version History

- Revision 1, 2017-11-10 (Daniel Rakos)
 - Internal revisions

VK_EXT_filter_cubic

Name String

`VK_EXT_filter_cubic`

Extension Type

Device extension

Registered Extension Number

171

Revision

3

Ratification Status

Ratified

Extension and Version Dependencies

None

Contact

- Matthew Netsch [[GitHub](#)][mnetsch](#)

Other Extension Metadata

Last Modified Date

2019-12-13

Contributors

- Bill Licea-Kane, Qualcomm Technologies, Inc.
- Andrew Garrard, Samsung
- Daniel Koch, NVIDIA
- Donald Scorgie, Imagination Technologies
- Graeme Leese, Broadcom
- Jan-Harald Fredriksen, ARM
- Jeff Leger, Qualcomm Technologies, Inc.
- Tobias Hector, AMD
- Tom Olson, ARM
- Stuart Smith, Imagination Technologies

Description

`VK_EXT_filter_cubic` extends `VK_IMG_filter_cubic`.

It documents cubic filtering of other image view types. It adds new structures that **can** be added to the `pNext` chain of `VkPhysicalDeviceImageFormatInfo2` and `VkImageFormatProperties2` that **can** be used to determine which image types and which image view types support cubic filtering.

New Structures

- Extending `VkImageFormatProperties2`:
 - `VkFilterCubicImageViewImageFormatPropertiesEXT`
- Extending `VkPhysicalDeviceImageFormatInfo2`:
 - `VkPhysicalDeviceImageViewImageFormatInfoEXT`

New Enum Constants

- `VK_EXT_FILTER_CUBIC_EXTENSION_NAME`
- `VK_EXT_FILTER_CUBIC_SPEC_VERSION`
- Extending `VkFilter`:

- `VK_FILTER_CUBIC_EXT`
- Extending `VkFormatFeatureFlagBits`:
 - `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_CUBIC_BIT_EXT`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_FILTER_CUBIC_IMAGE_VIEW_IMAGE_FORMAT_PROPERTIES_EXT`
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_VIEW_IMAGE_FORMAT_INFO_EXT`

Version History

- Revision 3, 2019-12-13 (wwlk)
 - Delete requirement to cubic filter the formats `USCALED_PACKED32`, `SSCALED_PACKED32`, `UINT_PACK32`, and `SINT_PACK32` (cut/paste error)
- Revision 2, 2019-06-05 (wwlk)
 - Clarify 1D optional
- Revision 1, 2019-01-24 (wwlk)
 - Initial version

VK_EXT_fragment_shader_interlock

Name String

`VK_EXT_fragment_shader_interlock`

Extension Type

Device extension

Registered Extension Number

252

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_get_physical_device_properties2`

or

[Vulkan Version 1.1](#)

SPIR-V Dependencies

- [SPV_EXT_fragment_shader_interlock](#)

Contact

- Piers Daniell [\[GitHub\]](#)[pdaniell-nv](#)

Other Extension Metadata

Last Modified Date

2019-05-02

Interactions and External Dependencies

- This extension provides API support for `GL_ARB_fragment_shader_interlock`

Contributors

- Daniel Koch, NVIDIA
- Graeme Leese, Broadcom
- Jan-Harald Fredriksen, Arm
- Faith Ekstrand, Intel
- Jeff Bolz, NVIDIA
- Ruihao Zhang, Qualcomm
- Slawomir Grajewski, Intel
- Spencer Fricke, Samsung

Description

This extension adds support for the `FragmentShaderPixelInterlockEXT`, `FragmentShaderSampleInterlockEXT`, and `FragmentShaderShadingRateInterlockEXT` capabilities from the `SPV_EXT_fragment_shader_interlock` extension to Vulkan.

Enabling these capabilities provides a critical section for fragment shaders to avoid overlapping pixels being processed at the same time, and certain guarantees about the ordering of fragment shader invocations of fragments of overlapping pixels.

This extension can be useful for algorithms that need to access per-pixel data structures via shader loads and stores. Algorithms using this extension can access per-pixel data structures in critical sections without other invocations accessing the same per-pixel data. Additionally, the ordering guarantees are useful for cases where the API ordering of fragments is meaningful. For example, applications may be able to execute programmable blending operations in the fragment shader, where the destination buffer is read via image loads and the final value is written via image stores.

New Structures

- Extending `VkPhysicalDeviceFeatures2`, `VkDeviceCreateInfo`:
 - `VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT`

New Enum Constants

- `VK_EXT_FRAGMENT_SHADER_INTERLOCK_EXTENSION_NAME`
- `VK_EXT_FRAGMENT_SHADER_INTERLOCK_SPEC_VERSION`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_FRAGMENT_SHADER_INTERLOCK_FEATURES_EXT`

New SPIR-V Capabilities

- `FragmentShaderInterlockEXT`
- `FragmentShaderPixelInterlockEXT`
- `FragmentShaderShadingRateInterlockEXT`

Version History

- Revision 1, 2019-05-24 (Piers Daniell)
 - Internal revisions

VK_EXT_hdr_metadata

Name String

`VK_EXT_hdr_metadata`

Extension Type

Device extension

Registered Extension Number

106

Revision

3

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_swapchain](#)

Contact

- Courtney Goeltzenleuchter [\[GitHub\]](#)`courtney-g`

Other Extension Metadata

Last Modified Date

2024-03-26

IP Status

No known IP claims.

Contributors

- Courtney Goeltzenleuchter, Google
- Sebastian Wick, Red Hat Inc.
- Tobias Hector, AMD

Description

This extension defines two new structures and a function to assign SMPTE (the Society of Motion Picture and Television Engineers) 2086 metadata and CTA (Consumer Technology Association) 861.3 metadata to a swapchain.

SMPTE 2086 metadata defines the color volume of the display on which the content was optimized for viewing and includes the color primaries, white point, and luminance range. When such content is reproduced on another display, this metadata can be used by the presentation engine to improve processing of images. For instance, values in the image can first be clamped to the color volume described in the metadata, and then what remains can be remapped to the color volume of the presentation engine.

CTA 861.3 metadata additionally includes the maximum intended luminance for the content and the maximum average light level across frames.

This extension does not define exactly how this metadata is used, however, it simply provides a mechanism to provide it to the presentation engine. Presentation engines may process the image based on the metadata before displaying it, resulting in the image being modified outside of Vulkan. For example, the clamping of colors in the image to the color volume may change those values in the image itself.

The metadata does not override or otherwise influence the color space and color encoding.

New Commands

- [vkSetHdrMetadataEXT](#)

New Structures

- [VkHdrMetadataEXT](#)
- [VkXYColorEXT](#)

New Enum Constants

- [VK_EXT_HDR_METADATA_EXTENSION_NAME](#)
- [VK_EXT_HDR_METADATA_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_HDR_METADATA_EXT](#)

Issues

1) Do we need a query function for the currently specified metadata?

No, Vulkan does not provide queries for state that the application can track on its own.

2) Should we specify default metadata if not specified by the application?

No, the metadata is optional and the absence of the metadata is well-defined.

Version History

- Revision 1, 2016-12-27 (Courtney Goeltzenleuchter)
 - Initial version
- Revision 2, 2018-12-19 (Courtney Goeltzenleuchter)
 - Correct implicit validity for VkHdrMetadataEXT structure
- Revision 3, 2024-03-26 (Tobias Hector & Sebastian Wick)
 - Clarifications and removal of erroneous "reference monitor" term

VK_EXT_headless_surface

Name String

VK_EXT_headless_surface

Extension Type

Instance extension

Registered Extension Number

257

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_surface](#)

Contact

- Lisa Wu [\[GitHub\]chengtianww](#)

Other Extension Metadata

Last Modified Date

2019-03-21

IP Status

No known IP claims.

Contributors

- Ray Smith, Arm

Description

The [VK_EXT_headless_surface](#) extension is an instance extension. It provides a mechanism to create [VkSurfaceKHR](#) objects independently of any window system or display device. The presentation

operation for a swapchain created from a headless surface is by default a no-op, resulting in no externally-visible result.

Because there is no real presentation target, future extensions can layer on top of the headless surface to introduce arbitrary or customizable sets of restrictions or features. These could include features like saving to a file or restrictions to emulate a particular presentation target.

This functionality is expected to be useful for application and driver development because it allows any platform to expose an arbitrary or customizable set of restrictions and features of a presentation engine. This makes it a useful portable test target for applications targeting a wide range of presentation engines where the actual target presentation engines might be scarce, unavailable or otherwise undesirable or inconvenient to use for general Vulkan application development.

New Commands

- [vkCreateHeadlessSurfaceEXT](#)

New Structures

- [VkHeadlessSurfaceCreateInfoEXT](#)

New Bitmasks

- [VkHeadlessSurfaceCreateFlagsEXT](#)

New Enum Constants

- [VK_EXT_HEADLESS_SURFACE_EXTENSION_NAME](#)
- [VK_EXT_HEADLESS_SURFACE_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_HEADLESS_SURFACE_CREATE_INFO_EXT](#)

Version History

- Revision 1, 2019-03-21 (Ray Smith)
 - Initial draft

VK_EXT_image_drm_format_modifier

Name String

[VK_EXT_image_drm_format_modifier](#)

Extension Type

Device extension

Registered Extension Number

159

Revision

2

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_bind_memory2`
and
`VK_KHR_get_physical_device_properties2`
and
`VK_KHR_sampler_ycbcr_conversion`
or
[Vulkan Version 1.1](#)
and
`VK_KHR_image_format_list`
or
[Vulkan Version 1.2](#)

API Interactions

- Interacts with `VK_VERSION_1_3`
- Interacts with `VK_KHR_format_feature_flags2`

Contact

- Lina Versace [\[GitHub\]](#)[linyaa-kiwi](#)

Other Extension Metadata

Last Modified Date

2021-09-30

IP Status

No known IP claims.

Contributors

- Antoine Labour, Google
- Bas Nieuwenhuizen, Google
- Lina Versace, Google
- James Jones, NVIDIA
- Faith Ekstrand, Intel
- Jörg Wagner, ARM
- Kristian Høgsberg Kristensen, Google
- Ray Smith, ARM

Description

This extension provides the ability to use *DRM format modifiers* with images, enabling Vulkan to better integrate with the Linux ecosystem of graphics, video, and display APIs.

Its functionality closely overlaps with `EGL_EXT_image_dma_buf_import_modifiers`² and `EGL_MESA_image_dma_buf_export`³. Unlike the EGL extensions, this extension does not require the use of a specific handle type (such as a `dma_buf`) for external memory and provides more explicit control of image creation.

Introduction to DRM Format Modifiers

A *DRM format modifier* is a 64-bit, vendor-prefixed, semi-opaque unsigned integer. Most *modifiers* represent a concrete, vendor-specific tiling format for images. Some exceptions are `DRM_FORMAT_MOD_LINEAR` (which is not vendor-specific); `DRM_FORMAT_MOD_NONE` (which is an alias of `DRM_FORMAT_MOD_LINEAR` due to historical accident); and `DRM_FORMAT_MOD_INVALID` (which does not represent a tiling format). The *modifier*'s vendor prefix consists of the 8 most significant bits. The canonical list of *modifiers* and vendor prefixes is found in `drm_fourcc.h` in the Linux kernel source. The other dominant source of *modifiers* are vendor kernel trees.

One goal of *modifiers* in the Linux ecosystem is to enumerate for each vendor a reasonably sized set of tiling formats that are appropriate for images shared across processes, APIs, and/or devices, where each participating component may possibly be from different vendors. A non-goal is to enumerate all tiling formats supported by all vendors. Some tiling formats used internally by vendors are inappropriate for sharing; no *modifiers* should be assigned to such tiling formats.

Modifier values typically do not *describe* memory layouts. More precisely, a *modifier*'s lower 56 bits usually have no structure. Instead, *modifiers name* memory layouts; they name a small set of vendor-preferred layouts for image sharing. As a consequence, in each vendor namespace the modifier values are often sequentially allocated starting at 1.

Each *modifier* is usually supported by a single vendor and its name matches the pattern `{VENDOR}_FORMAT_MOD_*` or `DRM_FORMAT_MOD_{VENDOR}_*`. Examples are `I915_FORMAT_MOD_X_TILED` and `DRM_FORMAT_MOD_BROADCOM_VC4_T_TILED`. An exception is `DRM_FORMAT_MOD_LINEAR`, which is supported by most vendors.

Many APIs in Linux use *modifiers* to negotiate and specify the memory layout of shared images. For example, a Wayland compositor and Wayland client may, by relaying *modifiers* over the Wayland protocol `zwp_linux_dmabuf_v1`, negotiate a vendor-specific tiling format for a shared `wl_buffer`. The client may allocate the underlying memory for the `wl_buffer` with GBM, providing the chosen *modifier* to `gbm_bo_create_with_modifiers`. The client may then import the `wl_buffer` into Vulkan for producing image content, providing the resource's `dma_buf` to `VkImportMemoryFdInfoKHR` and its *modifier* to `VkImageDrmFormatModifierExplicitCreateInfoEXT`. The compositor may then import the `wl_buffer` into OpenGL for sampling, providing the resource's `dma_buf` and *modifier* to `eglCreateImage`. The compositor may also bypass OpenGL and submit the `wl_buffer` directly to the kernel's display API, providing the `dma_buf` and *modifier* through `drm_mode_fb_cmd2`.

Format Translation

Modifier-capable APIs often pair *modifiers* with DRM formats, which are defined in `drm_fourcc.h`.

However, `VK_EXT_image_drm_format_modifier` uses `VkFormat` instead of DRM formats. The application must convert between `VkFormat` and DRM format when it sends or receives a DRM format to or from an external API.

The mapping from `VkFormat` to DRM format is lossy. Therefore, when receiving a DRM format from an external API, often the application must use information from the external API to accurately map the DRM format to a `VkFormat`. For example, DRM formats do not distinguish between RGB and sRGB (as of 2018-03-28); external information is required to identify the image's color space.

The mapping between `VkFormat` and DRM format is also incomplete. For some DRM formats there exist no corresponding Vulkan format, and for some Vulkan formats there exist no corresponding DRM format.

Usage Patterns

Three primary usage patterns are intended for this extension:

- **Negotiation.** The application negotiates with *modifier*-aware, external components to determine sets of image creation parameters supported among all components.

In the Linux ecosystem, the negotiation usually assumes the image is a 2D, single-sampled, non-mipmapped, non-array image; this extension permits that assumption but does not require it. The result of the negotiation usually resembles a set of tuples such as *(drmFormat, drmFormatModifier)*, where each participating component supports all tuples in the set.

Many details of this negotiation - such as the protocol used during negotiation, the set of image creation parameters expressible in the protocol, and how the protocol chooses which process and which API will create the image - are outside the scope of this specification.

In this extension, `vkGetPhysicalDeviceFormatProperties2` with `VkDrmFormatModifierPropertiesListEXT` serves a primary role during the negotiation, and `vkGetPhysicalDeviceImageFormatProperties2` with `VkPhysicalDeviceImageDrmFormatModifierInfoEXT` serves a secondary role.

- **Import.** The application imports an image with a *modifier*.

In this pattern, the application receives from an external source the image's memory and its creation parameters, which are often the result of the negotiation described above. Some image creation parameters are implicitly defined by the external source; for example, `VK_IMAGE_TYPE_2D` is often assumed. Some image creation parameters are usually explicit, such as the image's *format*, *drmFormatModifier*, and *extent*; and each plane's *offset* and *rowPitch*.

Before creating the image, the application first verifies that the physical device supports the received creation parameters by querying `vkGetPhysicalDeviceFormatProperties2` with `VkDrmFormatModifierPropertiesListEXT` and `vkGetPhysicalDeviceImageFormatProperties2` with `VkPhysicalDeviceImageDrmFormatModifierInfoEXT`. Then the application creates the image by chaining `VkImageDrmFormatModifierExplicitCreateInfoEXT` and `VkExternalMemoryImageCreateInfo` onto `VkImageCreateInfo`.

- **Export.** The application creates an image and allocates its memory. Then the application

exports to *modifier*-aware consumers the image's memory handles; its creation parameters; its *modifier*; and the *offset*, *size*, and *rowPitch* of each *memory plane*.

In this pattern, the Vulkan device is the authority for the image; it is the allocator of the image's memory and the decider of the image's creation parameters. When choosing the image's creation parameters, the application usually chooses a tuple (*format*, *drmFormatModifier*) from the result of the negotiation described above. The negotiation's result often contains multiple tuples that share the same format but differ in their *modifier*. In this case, the application should defer the choice of the image's *modifier* to the Vulkan implementation by providing all such *modifiers* to `VkImageDrmFormatModifierListCreateInfoEXT::pDrmFormatModifiers`; and the implementation should choose from `pDrmFormatModifiers` the optimal *modifier* in consideration with the other image parameters.

The application creates the image by chaining `VkImageDrmFormatModifierListCreateInfoEXT` and `VkExternalMemoryImageCreateInfo` onto `VkImageCreateInfo`. The protocol and APIs by which the application will share the image with external consumers will likely determine the value of `VkExternalMemoryImageCreateInfo::handleTypes`. The implementation chooses for the image an optimal *modifier* from `VkImageDrmFormatModifierListCreateInfoEXT::pDrmFormatModifiers`. The application then queries the implementation-chosen *modifier* with `vkGetImageDrmFormatModifierPropertiesEXT`, and queries the memory layout of each plane with `vkGetImageSubresourceLayout`.

The application then allocates the image's memory with `VkMemoryAllocateInfo`, adding chained extending structures for external memory; binds it to the image; and exports the memory, for example, with `vkGetMemoryFdKHR`.

Finally, the application sends the image's creation parameters, its *modifier*, its per-plane memory layout, and the exported memory handle to the external consumers. The details of how the application transmits this information to external consumers is outside the scope of this specification.

Prior Art

Extension `EGL_EXT_image_dma_buf_import`¹ introduced the ability to create an `EGLImage` by importing for each plane a `dma_buf`, `offset`, and `row pitch`.

Later, extension `EGL_EXT_image_dma_buf_import_modifiers`² introduced the ability to query which combination of formats and *modifiers* the implementation supports and to specify *modifiers* during creation of the `EGLImage`.

Extension `EGL_MESA_image_dma_buf_export`³ is the inverse of `EGL_EXT_image_dma_buf_import_modifiers`.

The Linux kernel modesetting API (KMS), when configuring the display's framebuffer with `struct drm_mode_fb_cmd2`⁴, allows one to specify the framebuffer's *modifier* as well as a per-plane memory handle, `offset`, and `row pitch`.

GBM, a graphics buffer manager for Linux, allows creation of a `gbm_bo` (that is, a graphics *buffer object*) by importing data similar to that in `EGL_EXT_image_dma_buf_import_modifiers`¹; and symmetrically allows exporting the same data from the `gbm_bo`. See the references to *modifier* and *plane* in `gbm.h`⁵.

New Commands

- [vkGetImageDrmFormatModifierPropertiesEXT](#)

New Structures

- [VkDrmFormatModifierPropertiesEXT](#)
- [VkImageDrmFormatModifierPropertiesEXT](#)
- Extending [VkFormatProperties2](#):
 - [VkDrmFormatModifierPropertiesListEXT](#)
- Extending [VkImageCreateInfo](#):
 - [VkImageDrmFormatModifierExplicitCreateInfoEXT](#)
 - [VkImageDrmFormatModifierListCreateInfoEXT](#)
- Extending [VkPhysicalDeviceImageFormatInfo2](#):
 - [VkPhysicalDeviceImageDrmFormatModifierInfoEXT](#)

If [VK_KHR_format_feature_flags2](#) or [Vulkan Version 1.3](#) is supported:

- [VkDrmFormatModifierProperties2EXT](#)
- Extending [VkFormatProperties2](#):
 - [VkDrmFormatModifierPropertiesList2EXT](#)

New Enum Constants

- [VK_EXT_IMAGE_DRM_FORMAT_MODIFIER_EXTENSION_NAME](#)
- [VK_EXT_IMAGE_DRM_FORMAT_MODIFIER_SPEC_VERSION](#)
- Extending [VkImageAspectFlagBits](#):
 - [VK_IMAGE_ASPECT_MEMORY_PLANE_0_BIT_EXT](#)
 - [VK_IMAGE_ASPECT_MEMORY_PLANE_1_BIT_EXT](#)
 - [VK_IMAGE_ASPECT_MEMORY_PLANE_2_BIT_EXT](#)
 - [VK_IMAGE_ASPECT_MEMORY_PLANE_3_BIT_EXT](#)
- Extending [VkImageTiling](#):
 - [VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT](#)
- Extending [VkResult](#):
 - [VK_ERROR_INVALID_DRM_FORMAT_MODIFIER_PLANE_LAYOUT_EXT](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_DRM_FORMAT_MODIFIER_PROPERTIES_LIST_EXT](#)
 - [VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_EXPLICIT_CREATE_INFO_EXT](#)
 - [VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_LIST_CREATE_INFO_EXT](#)
 - [VK_STRUCTURE_TYPE_IMAGE_DRM_FORMAT_MODIFIER_PROPERTIES_EXT](#)

- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_DRM_FORMAT_MODIFIER_INFO_EXT`

If `VK_KHR_format_feature_flags2` or Vulkan Version 1.3 is supported:

- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_DRM_FORMAT_MODIFIER_PROPERTIES_LIST_2_EXT`

Issues

1) Should this extension define a single DRM format modifier per `VkImage`? Or define one per plane?

+

RESOLVED: There exists a single DRM format modifier per `VkImage`.

DISCUSSION: Prior art, such as `EGL_EXT_image_dma_buf_import_modifiers`², `struct drm_mode_fb_cmd2`⁴, and `struct gbm_import_fd_modifier_data`⁵, allows defining one *modifier* per plane. However, developers of the GBM and kernel APIs concede it was a mistake. Beginning in Linux 4.10, the kernel requires that the application provide the same DRM format *modifier* for each plane. (See Linux commit [bae781b259269590109e8a4a8227331362b88212](#)). And GBM provides an entry point, `gbm_bo_get_modifier`, for querying the *modifier* of the image but does not provide one to query the modifier of individual planes.

2) When creating an image with `VkImageDrmFormatModifierExplicitCreateInfoEXT`, which is typically used when *importing* an image, should the application explicitly provide the size of each plane?

+

RESOLVED: No. The application **must** not provide the size. To enforce this, the API requires that `VkImageDrmFormatModifierExplicitCreateInfoEXT::pPlaneLayouts->size` **must** be 0.

DISCUSSION: Prior art, such as `EGL_EXT_image_dma_buf_import_modifiers`², `struct drm_mode_fb_cmd2`⁴, and `struct gbm_import_fd_modifier_data`⁵, omits from the API the size of each plane. Instead, the APIs infer each plane's size from the import parameters, which include the image's pixel format and a `dma_buf`, offset, and row pitch for each plane.

However, Vulkan differs from EGL and GBM with regards to image creation in the following ways:

Differences in Image Creation

- **Undedicated allocation by default.** When importing or exporting a set of `dma_bufs` as an `EGLImage` or `gbm_bo`, common practice mandates that each `dma_buf`'s memory be dedicated (in the sense of `VK_KHR_dedicated_allocation`) to the image (though not necessarily dedicated to a single plane). In particular, neither the GBM documentation nor the EGL extension specifications explicitly state this requirement, but in light of common practice this is likely due to under-specification rather than intentional omission. In contrast, `VK_EXT_image_drm_format_modifier` permits, but does not require, the implementation to require dedicated allocations for images created with `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`.
- **Separation of image creation and memory allocation.** When importing a set of `dma_bufs` as an `EGLImage` or `gbm_bo`, EGL and GBM create the image resource and bind it to memory (the

dma_bufs) simultaneously. This allows EGL and GBM to query each dma_buf's size during image creation. In Vulkan, image creation and memory allocation are independent unless a dedicated allocation is used (as in [VK_KHR_dedicated_allocation](#)). Therefore, without requiring dedicated allocation, Vulkan cannot query the size of each dma_buf (or other external handle) when calculating the image's memory layout. Even if dedication allocation were required, Vulkan cannot calculate the image's memory layout until after the image is bound to its dma_bufs.

The above differences complicate the potential inference of plane size in Vulkan. Consider the following problematic cases:

Problematic Plane Size Calculations

- **Padding.** Some plane of the image may require implementation-dependent padding.
- **Metadata.** For some *modifiers*, the image may have a metadata plane which requires a non-trivial calculation to determine its size.
- **Mipmapped, array, and 3D images.** The implementation may support [VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT](#) for images whose *mipLevels*, *arrayLayers*, or *depth* is greater than 1. For such images with certain *modifiers*, the calculation of each plane's size may be non-trivial.

However, an application-provided plane size solves none of the above problems.

For simplicity, consider an external image with a single memory plane. The implementation is obviously capable calculating the image's size when its tiling is [VK_IMAGE_TILING_OPTIMAL](#). Likewise, any reasonable implementation is capable of calculating the image's size when its tiling uses a supported *modifier*.

Suppose that the external image's size is smaller than the implementation-calculated size. If the application provided the external image's size to [vkCreateImage](#), the implementation would observe the mismatched size and recognize its inability to comprehend the external image's layout (unless the implementation used the application-provided size to select a refinement of the tiling layout indicated by the *modifier*, which is strongly discouraged). The implementation would observe the conflict, and reject image creation with [VK_ERROR_INVALID_DRM_FORMAT_MODIFIER_PLANE_LAYOUT_EXT](#). On the other hand, if the application did not provide the external image's size to [vkCreateImage](#), then the application would observe after calling [vkGetImageMemoryRequirements](#) that the external image's size is less than the size required by the implementation. The application would observe the conflict and refuse to bind the [VkImage](#) to the external memory. In both cases, the result is explicit failure.

Suppose that the external image's size is larger than the implementation-calculated size. If the application provided the external image's size to [vkCreateImage](#), for reasons similar to above the implementation would observe the mismatched size and recognize its inability to comprehend the image data residing in the extra size. The implementation, however, must assume that image data resides in the entire size provided by the application. The implementation would observe the conflict and reject image creation with [VK_ERROR_INVALID_DRM_FORMAT_MODIFIER_PLANE_LAYOUT_EXT](#). On the other hand, if the application did not provide the external image's size to [vkCreateImage](#), then the application would observe after calling [vkGetImageMemoryRequirements](#) that the external image's size is larger than the implementation-usable size. The application would observe the

conflict and refuse to bind the `VkImage` to the external memory. In both cases, the result is explicit failure.

Therefore, an application-provided size provides no benefit, and this extension should not require it. This decision renders `VkSubresourceLayout::size` an unused field during image creation, and thus introduces a risk that implementations may require applications to submit sideband creation parameters in the unused field. To prevent implementations from relying on sideband data, this extension *requires* the application to set `size` to 0.

References

1. `EGL_EXT_image_dma_buf_import`
2. `EGL_EXT_image_dma_buf_import_modifiers`
3. `EGL_MESA_image_dma_buf_export`
4. `struct drm_mode_fb_cmd2`
5. `gbm.h`

Version History

- Revision 1, 2018-08-29 (Lina Versace)
 - First stable revision
- Revision 2, 2021-09-30 (Jon Leech)
 - Add interaction with `VK_KHR_format_feature_flags2` to `vk.xml`

VK_EXT_image_robustness

Name String

`VK_EXT_image_robustness`

Extension Type

Device extension

Registered Extension Number

336

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_get_physical_device_properties2`

or

[Vulkan Version 1.1](#)

Deprecation State

- *Promoted* to Vulkan 1.3

Contact

- Graeme Leese [\[GitHub\]](#)gnl21

Other Extension Metadata

Last Modified Date

2020-04-27

IP Status

No known IP claims.

Contributors

- Graeme Leese, Broadcom
- Jan-Harald Fredriksen, ARM
- Jeff Bolz, NVIDIA
- Spencer Fricke, Samsung
- Courtney Goeltzenleuchter, Google
- Slawomir Cygan, Intel

Description

This extension adds stricter requirements for how out of bounds reads from images are handled. Rather than returning undefined values, most out of bounds reads return R, G, and B values of zero and alpha values of either zero or one. Components not present in the image format may be set to zero or to values based on the format as described in [Component Substitution](#).

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceImageRobustnessFeaturesEXT](#)

New Enum Constants

- [VK_EXT_IMAGE_ROBUSTNESS_EXTENSION_NAME](#)
- [VK_EXT_IMAGE_ROBUSTNESS_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_IMAGE_ROBUSTNESS_FEATURES_EXT](#)

Promotion to Vulkan 1.3

Vulkan APIs in this extension are included in core Vulkan 1.3, with the EXT suffix omitted. External interactions defined by this extension, such as SPIR-V token names, retain their original names. The original Vulkan API names are still available as aliases of the core functionality.

Issues

1. How does this extension differ from VK_EXT_robustness2?

The guarantees provided by this extension are a subset of those provided by the robustImageAccess2 feature of VK_EXT_robustness2. Where this extension allows return values of (0, 0, 0, 0) or (0, 0, 0, 1), robustImageAccess2 requires that a particular value dependent on the image format be returned. This extension provides no guarantees about the values returned for an access to an invalid Lod.

Examples

None.

Version History

- Revision 1, 2020-04-27 (Graeme Leese)
- Initial draft

VK_EXT_layer_settings

Name String

VK_EXT_layer_settings

Extension Type

Instance extension

Registered Extension Number

497

Revision

2

Ratification Status

Ratified

Extension and Version Dependencies

None

Contact

- Christophe Riccio [\[GitHub\]](#)christophe

Extension Proposal

[VK_EXT_layer_settings](#)

Other Extension Metadata

Last Modified Date

2023-09-23

IP Status

No known IP claims.

Contributors

- Christophe Riccio, LunarG
- Mark Lobodzinski, LunarG
- Charles Giessen, LunarG
- Spencer Fricke, LunarG
- Juan Ramos, LunarG
- Daniel Rakos, RasterGrid
- Shahbaz Youssefi, Google
- Lina Versace, Google
- Bill Hollings, The Brenwill Workshop
- Jon Leech, Khronos
- Tom Olson, Arm

Description

This extension provides a mechanism for configuring programmatically through the Vulkan API the behavior of layers.

This extension provides the [VkLayerSettingsCreateInfoEXT](#) structure that can be included in the [pNext](#) chain of the [VkInstanceCreateInfo](#) structure passed as the [pCreateInfo](#) parameter of [vkCreateInstance](#).

The structure contains an array of [VkLayerSettingEXT](#) structure values that configure specific features of layers.

NOTE

The [VK_EXT_layer_settings](#) extension subsumes all the functionality provided in the [VK_EXT_validation_flags](#) extension and the [VK_EXT_validation_features](#) extension.

New Structures

- [VkLayerSettingEXT](#)
- Extending [VkInstanceCreateInfo](#):
 - [VkLayerSettingsCreateInfoEXT](#)

New Enums

- [VkLayerSettingTypeEXT](#)

New Enum Constants

- [VK_EXT_LAYER_SETTINGS_EXTENSION_NAME](#)

- `VK_EXT_LAYER_SETTINGS_SPEC_VERSION`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_LAYER_SETTINGS_CREATE_INFO_EXT`

Example

One example usage of `VK_EXT_layer_settings` is as implemented by the Vulkan Profiles layer.

It allows the profiles layer tests used by the profiles layer C.I. to programmatically configure the layer for each test without affecting the C.I. environment, allowing to run multiple tests concurrently.

```
const char* profile_file_data = JSON_TEST_FILES_PATH "VP_KHR_roadmap_2022.json";
const char* profile_name_data = "VP_KHR_roadmap_2022";
VkBool32 emulate_portability_data = VK_TRUE;
const char* simulate_capabilities[] = {
    "SIMULATE_API_VERSION_BIT",
    "SIMULATE_FEATURES_BIT",
    "SIMULATE_PROPERTIES_BIT",
    "SIMULATE_EXTENSIONS_BIT",
    "SIMULATE_FORMATS_BIT",
    "SIMULATE_QUEUE_FAMILY_PROPERTIES_BIT"
};
const char* debug_reports[] = {
    "DEBUG_REPORT_ERROR_BIT",
    "DEBUG_REPORT_WARNING_BIT",
    "DEBUG_REPORT_NOTIFICATION_BIT",
    "DEBUG_REPORT_DEBUG_BIT"
};

const VkLayerSettingEXT settings[] = {
    {kLayerName, kLayerSettingsProfileFile, VK_LAYER_SETTING_TYPE_STRING_EXT, 1,
    &profile_file_data},
    {kLayerName, kLayerSettingsProfileName, VK_LAYER_SETTING_TYPE_STRING_EXT, 1,
    &profile_name_data},
    {kLayerName, kLayerSettingsEmulatePortability, VK_LAYER_SETTING_TYPE_BOOL32_EXT,
    1, &emulate_portability_data},
    {kLayerName, kLayerSettingsSimulateCapabilities,
    VK_LAYER_SETTING_TYPE_STRING_EXT,
    static_cast<uint32_t>(std::size(simulate_capabilities)),
    simulate_capabilities},
    {kLayerName, kLayerSettingsDebugReports, VK_LAYER_SETTING_TYPE_STRING_EXT,
    static_cast<uint32_t>(std::size(debug_reports)), debug_reports}
};

const VkLayerSettingsCreateInfoEXT layer_settings_create_info{
    VK_STRUCTURE_TYPE_LAYER_SETTINGS_CREATE_INFO_EXT, nullptr,
    static_cast<uint32_t>(std::size(settings)), settings};
```

```
VkInstanceCreateInfo inst_create_info = {};  
...  
inst_create_info.pNext = &layer_settings_create_info;  
vkCreateInstance(&inst_create_info, nullptr, &instances);
```

Issues

- How should application developers figure out the list of available settings?

This extension does not provide a reflection API for layer settings. Layer settings are described in each layer JSON manifest and the documentation of each layer which implements this extension.

Version History

- Revision 1, 2020-06-17 (Mark Lobodzinski)
 - Initial revision for Validation layer internal usages
- Revision 2, 2023-09-26 (Christophe Riccio)
 - Refactor APIs for any layer usages and public release

VK_EXT_memory_budget

Name String

VK_EXT_memory_budget

Extension Type

Device extension

Registered Extension Number

238

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

Vulkan Version 1.1

Contact

- Jeff Bolz [[GitHub](#)]jeffbolznv

Other Extension Metadata

Last Modified Date

2018-10-08

Contributors

- Jeff Bolz, NVIDIA
- Jeff Juliano, NVIDIA

Description

While running a Vulkan application, other processes on the machine might also be attempting to use the same device memory, which can pose problems. This extension adds support for querying the amount of memory used and the total memory budget for a memory heap. The values returned by this query are implementation-dependent and can depend on a variety of factors including operating system and system load.

The [VkPhysicalDeviceMemoryBudgetPropertiesEXT::heapBudget](#) values can be used as a guideline for how much total memory from each heap the **current process** can use at any given time, before allocations may start failing or causing performance degradation. The values may change based on other activity in the system that is outside the scope and control of the Vulkan implementation.

The [VkPhysicalDeviceMemoryBudgetPropertiesEXT::heapUsage](#) will display the **current process** estimated heap usage.

With this information, the idea is for an application at some interval (once per frame, per few seconds, etc) to query [heapBudget](#) and [heapUsage](#). From here the application can notice if it is over budget and decide how it wants to handle the memory situation (free it, move to host memory, changing mipmap levels, etc). This extension is designed to be used in concert with [VK_EXT_memory_priority](#) to help with this part of memory management.

New Structures

- Extending [VkPhysicalDeviceMemoryProperties2](#):
 - [VkPhysicalDeviceMemoryBudgetPropertiesEXT](#)

New Enum Constants

- [VK_EXT_MEMORY_BUDGET_EXTENSION_NAME](#)
- [VK_EXT_MEMORY_BUDGET_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_MEMORY_BUDGET_PROPERTIES_EXT](#)

Version History

- Revision 1, 2018-10-08 (Jeff Bolz)
 - Initial revision

VK_EXT_pci_bus_info

Name String

VK_EXT_pci_bus_info

Extension Type

Device extension

Registered Extension Number

213

Revision

2

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

[Vulkan Version 1.1](#)

Contact

- Matthaeus G. Chajdas [\[GitHub\]anteru](#)

Other Extension Metadata

Last Modified Date

2018-12-10

IP Status

No known IP claims.

Contributors

- Matthaeus G. Chajdas, AMD
- Daniel Rakos, AMD

Description

This extension adds a new query to obtain PCI bus information about a physical device.

Not all physical devices have PCI bus information, either due to the device not being connected to the system through a PCI interface or due to platform specific restrictions and policies. Thus this extension is only expected to be supported by physical devices which can provide the information.

As a consequence, applications should always check for the presence of the extension string for each individual physical device for which they intend to issue the new query for and should not have any assumptions about the availability of the extension on any given platform.

New Structures

- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDevicePCIBusInfoPropertiesEXT](#)

New Enum Constants

- [VK_EXT_PCI_BUS_INFO_EXTENSION_NAME](#)
- [VK_EXT_PCI_BUS_INFO_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_PCI_BUS_INFO_PROPERTIES_EXT](#)

Version History

- Revision 2, 2018-12-10 (Daniel Rakos)
 - Changed all members of the new structure to have the uint32_t type
- Revision 1, 2018-10-11 (Daniel Rakos)
 - Initial revision

VK_EXT_post_depth_coverage

Name String

[VK_EXT_post_depth_coverage](#)

Extension Type

Device extension

Registered Extension Number

156

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

None

SPIR-V Dependencies

- [SPV_KHR_post_depth_coverage](#)

Contact

- Daniel Koch [\[GitHub\]dgkoch](#)

Other Extension Metadata

Last Modified Date

2017-07-17

Interactions and External Dependencies

- This extension provides API support for `GL_ARB_post_depth_coverage` and `GL_EXT_post_depth_coverage`

Contributors

- Jeff Bolz, NVIDIA

Description

This extension adds support for the following SPIR-V extension in Vulkan:

- `SPV_KHR_post_depth_coverage`

which allows the fragment shader to control whether values in the `SampleMask` built-in input variable reflect the coverage after early `depth` and `stencil` tests are applied.

This extension adds a new `PostDepthCoverage` execution mode under the `SampleMaskPostDepthCoverage` capability. When this mode is specified along with `EarlyFragmentTests`, the value of an input variable decorated with the `SampleMask` built-in reflects the coverage after the early fragment tests are applied. Otherwise, it reflects the coverage before the depth and stencil tests.

When using GLSL source-based shading languages, the `post_depth_coverage` layout qualifier from `GL_ARB_post_depth_coverage` or `GL_EXT_post_depth_coverage` maps to the `PostDepthCoverage` execution mode.

New Enum Constants

- `VK_EXT_POST_DEPTH_COVERAGE_EXTENSION_NAME`
- `VK_EXT_POST_DEPTH_COVERAGE_SPEC_VERSION`

New SPIR-V Capabilities

- `SampleMaskPostDepthCoverage`

Version History

- Revision 1, 2017-07-17 (Daniel Koch)
 - Internal revisions

VK_EXT_queue_family_foreign

Name String

`VK_EXT_queue_family_foreign`

Extension Type

Device extension

Registered Extension Number

127

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_external_memory](#)

or

[Vulkan Version 1.1](#)

Contact

- Lina Versace [\[GitHub\]](#)[linyaa-kiwi](#)

Other Extension Metadata

Last Modified Date

2017-11-01

IP Status

No known IP claims.

Contributors

- Lina Versace, Google
- James Jones, NVIDIA
- Faith Ekstrand, Intel
- Jesse Hall, Google
- Daniel Rakos, AMD
- Ray Smith, ARM

Description

This extension defines a special queue family, [VK_QUEUE_FAMILY_FOREIGN_EXT](#), which can be used to transfer ownership of resources backed by external memory to foreign, external queues. This is similar to [VK_QUEUE_FAMILY_EXTERNAL_KHR](#), defined in [VK_KHR_external_memory](#). The key differences between the two are:

- The queues represented by [VK_QUEUE_FAMILY_EXTERNAL_KHR](#) must share the same physical device and the same driver version as the current [VkInstance](#). [VK_QUEUE_FAMILY_FOREIGN_EXT](#) has no such restrictions. It can represent devices and drivers from other vendors, and can even represent non-Vulkan-capable devices.

- All resources backed by external memory support `VK_QUEUE_FAMILY_EXTERNAL_KHR`. Support for `VK_QUEUE_FAMILY_FOREIGN_EXT` is more restrictive.
- Applications should expect transitions to/from `VK_QUEUE_FAMILY_FOREIGN_EXT` to be more expensive than transitions to/from `VK_QUEUE_FAMILY_EXTERNAL_KHR`.

New Enum Constants

- `VK_EXT_QUEUE_FAMILY_FOREIGN_EXTENSION_NAME`
- `VK_EXT_QUEUE_FAMILY_FOREIGN_SPEC_VERSION`
- `VK_QUEUE_FAMILY_FOREIGN_EXT`

Version History

- Revision 1, 2017-11-01 (Lina Versace)
 - Squashed internal revisions

VK_EXT_robustness2

Name String

`VK_EXT_robustness2`

Extension Type

Device extension

Registered Extension Number

287

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_get_physical_device_properties2`

or

[Vulkan Version 1.1](#)

Deprecation State

- *Promoted* to `VK_KHR_robustness2` extension
 - **NOTE** The extension `VK_KHR_robustness2` is not supported for the API specification being generated

Contact

- Liam Middlebrook [\[GitHub\]liam-middlebrook](#)

Other Extension Metadata

Last Modified Date

2020-01-29

IP Status

No known IP claims.

Contributors

- Liam Middlebrook, NVIDIA
- Jeff Bolz, NVIDIA

Description

This extension adds stricter requirements for how out of bounds reads and writes are handled. Most accesses **must** be tightly bounds-checked, out of bounds writes **must** be discarded, out of bound reads **must** return zero. Rather than allowing multiple possible (0,0,0,x) vectors, the out of bounds values are treated as zero, and then missing components are inserted based on the format as described in [Component Substitution](#) and [vertex input attribute extraction](#).

These additional requirements **may** be expensive on some implementations, and should only be enabled when truly necessary.

This extension also adds support for “null descriptors”, where [VK_NULL_HANDLE](#) **can** be used instead of a valid handle. Accesses to null descriptors have well-defined behavior, and do not rely on robustness.

Promotion to [VK_KHR_robustness2](#)

All functionality in this extension is included in [VK_KHR_robustness2](#), with the suffix changed to KHR. The original type, enum, and command names are still available as aliases of the core functionality.

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceRobustness2FeaturesEXT](#)
- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDeviceRobustness2PropertiesEXT](#)

New Enum Constants

- [VK_EXT_ROBUSTNESS_2_EXTENSION_NAME](#)
- [VK_EXT_ROBUSTNESS_2_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_FEATURES_EXT](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ROBUSTNESS_2_PROPERTIES_EXT](#)

Issues

1. Why do [VkPhysicalDeviceRobustness2PropertiesEXT::robustUniformBufferAccessSizeAlignment](#) and [VkPhysicalDeviceRobustness2PropertiesEXT::robustStorageBufferAccessSizeAlignment](#) exist?

RESOLVED: Some implementations cannot efficiently tightly bounds-check all buffer accesses. Rather, the size of the bound range is padded to some power of two multiple, up to 256 bytes for uniform buffers and up to 4 bytes for storage buffers, and that padded size is bounds-checked. This is sufficient to implement D3D-like behavior, because D3D only allows binding whole uniform buffers or ranges that are a multiple of 256 bytes, and D3D raw and structured buffers only support 32-bit accesses.

Examples

None.

Version History

- Revision 1, 2019-11-01 (Jeff Bolz, Liam Middlebrook)
 - Initial draft

VK_EXT_sample_locations

Name String

[VK_EXT_sample_locations](#)

Extension Type

Device extension

Registered Extension Number

144

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_get_physical_device_properties2](#)

or

[Vulkan Version 1.1](#)

Contact

- Daniel Rakos [\[GitHub\]drakos-amd](#)

Other Extension Metadata

Last Modified Date

2017-08-02

Contributors

- Mais Alnasser, AMD
- Matthaeus G. Chajdas, AMD
- Maciej Jesionowski, AMD
- Daniel Rakos, AMD
- Slawomir Grajewski, Intel
- Jeff Bolz, NVIDIA
- Bill Licea-Kane, Qualcomm

Description

This extension allows an application to modify the locations of samples within a pixel used in rasterization. Additionally, it allows applications to specify different sample locations for each pixel in a group of adjacent pixels, which **can** increase antialiasing quality (particularly if a custom resolve shader is used that takes advantage of these different locations).

It is common for implementations to optimize the storage of depth values by storing values that **can** be used to reconstruct depth at each sample location, rather than storing separate depth values for each sample. For example, the depth values from a single triangle **may** be represented using plane equations. When the depth value for a sample is needed, it is automatically evaluated at the sample location. Modifying the sample locations causes the reconstruction to no longer evaluate the same depth values as when the samples were originally generated, thus the depth aspect of a depth/stencil attachment **must** be cleared before rendering to it using different sample locations.

Some implementations **may** need to evaluate depth image values while performing image layout transitions. To accommodate this, instances of the [VkSampleLocationsInfoEXT](#) structure **can** be specified for each situation where an explicit or automatic layout transition has to take place. [VkSampleLocationsInfoEXT](#) **can** be chained from [VkImageMemoryBarrier](#) structures to provide sample locations for layout transitions performed by [vkCmdWaitEvents](#) and [vkCmdPipelineBarrier](#) calls, and [VkRenderPassSampleLocationsBeginInfoEXT](#) **can** be chained from [VkRenderPassBeginInfo](#) to provide sample locations for layout transitions performed implicitly by a render pass instance.

New Commands

- [vkCmdSetSampleLocationsEXT](#)
- [vkGetPhysicalDeviceMultisamplePropertiesEXT](#)

New Structures

- [VkAttachmentSampleLocationsEXT](#)
- [VkMultisamplePropertiesEXT](#)
- [VkSampleLocationEXT](#)

- [VkSubpassSampleLocationsEXT](#)
- Extending [VkImageMemoryBarrier](#), [VkImageMemoryBarrier2](#):
 - [VkSampleLocationsInfoEXT](#)
- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDeviceSampleLocationsPropertiesEXT](#)
- Extending [VkPipelineMultisampleStateCreateInfo](#):
 - [VkPipelineSampleLocationsStateCreateInfoEXT](#)
- Extending [VkRenderPassBeginInfo](#):
 - [VkRenderPassSampleLocationsBeginInfoEXT](#)

New Enum Constants

- [VK_EXT_SAMPLE_LOCATIONS_EXTENSION_NAME](#)
- [VK_EXT_SAMPLE_LOCATIONS_SPEC_VERSION](#)
- Extending [VkDynamicState](#):
 - [VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT](#)
- Extending [VkImageCreateFlagBits](#):
 - [VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_MULTISAMPLE_PROPERTIES_EXT](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SAMPLE_LOCATIONS_PROPERTIES_EXT](#)
 - [VK_STRUCTURE_TYPE_PIPELINE_SAMPLE_LOCATIONS_STATE_CREATE_INFO_EXT](#)
 - [VK_STRUCTURE_TYPE_RENDER_PASS_SAMPLE_LOCATIONS_BEGIN_INFO_EXT](#)
 - [VK_STRUCTURE_TYPE_SAMPLE_LOCATIONS_INFO_EXT](#)

Issues

1) When using with Dynamic Rendering, is there a [VkRenderPassSampleLocationsBeginInfoEXT](#) equivalent struct

RESOLVED: No, there are no subpasses that need to have a sample location set.

Version History

- Revision 1, 2017-08-02 (Daniel Rakos)
 - Internal revisions

VK_EXT_shader_atomic_float

Name String

[VK_EXT_shader_atomic_float](#)

Extension Type

Device extension

Registered Extension Number

261

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_get_physical_device_properties2](#)

or

[Vulkan Version 1.1](#)

API Interactions

- Interacts [with](#) `VkPhysicalDeviceShaderAtomicFloatFeaturesEXT::sparseImageFloat32AtomicAdd`
- Interacts with `VkPhysicalDeviceShaderAtomicFloatFeaturesEXT::sparseImageFloat32Atomics`

SPIR-V Dependencies

- [SPV_EXT_shader_atomic_float_add](#)

Contact

- Vikram Kushwaha [\[GitHub\]](#)[vkushwaha-nv](#)

Other Extension Metadata

Last Modified Date

2020-07-15

IP Status

No known IP claims.

Interactions and External Dependencies

- This extension provides API support for [GL_EXT_shader_atomic_float](#)

Contributors

- Vikram Kushwaha, NVIDIA
- Jeff Bolz, NVIDIA

Description

This extension allows a shader to contain floating-point atomic operations on buffer, workgroup, and image memory. It also advertises the SPIR-V [AtomicFloat32AddEXT](#) and [AtomicFloat64AddEXT](#) capabilities that allows atomic addition on floating-points numbers. The supported operations include [OpAtomicFAddEXT](#), [OpAtomicExchange](#), [OpAtomicLoad](#) and [OpAtomicStore](#).

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceShaderAtomicFloatFeaturesEXT](#)

New Enum Constants

- [VK_EXT_SHADER_ATOMIC_FLOAT_EXTENSION_NAME](#)
- [VK_EXT_SHADER_ATOMIC_FLOAT_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_ATOMIC_FLOAT_FEATURES_EXT](#)

New SPIR-V Capabilities

- [AtomicFloat32AddEXT](#)
- [AtomicFloat64AddEXT](#)

Version History

- Revision 1, 2020-07-15 (Vikram Kushwaha)
 - Internal revisions

VK_EXT_shader_demote_to_helper_invocation

Name String

[VK_EXT_shader_demote_to_helper_invocation](#)

Extension Type

Device extension

Registered Extension Number

277

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_get_physical_device_properties2](#)

or

[Vulkan Version 1.1](#)

SPIR-V Dependencies

- [SPV_EXT_demote_to_helper_invocation](#)

Deprecation State

- *Promoted* to Vulkan 1.3

Contact

- Jeff Bolz [\[GitHub\]jeffbolznv](#)

Other Extension Metadata

Last Modified Date

2019-06-01

IP Status

No known IP claims.

Interactions and External Dependencies

- This extension provides API support for `GL_EXT_demote_to_helper_invocation`

Contributors

- Jeff Bolz, NVIDIA

Description

This extension adds Vulkan support for the `SPV_EXT_demote_to_helper_invocation` SPIR-V extension. That SPIR-V extension provides a new instruction `OpDemoteToHelperInvocationEXT` allowing shaders to “demote” a fragment shader invocation to behave like a helper invocation for its duration. The demoted invocation will have no further side effects and will not output to the framebuffer, but remains active and can participate in computing derivatives and in [group operations](#). This is a better match for the “discard” instruction in HLSL.

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceShaderDemoteToHelperInvocationFeaturesEXT](#)

New Enum Constants

- `VK_EXT_SHADER_DEMOTE_TO_HELPER_INVOCATION_EXTENSION_NAME`
- `VK_EXT_SHADER_DEMOTE_TO_HELPER_INVOCATION_SPEC_VERSION`
- Extending [VkStructureType](#):
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DEMOTE_TO_HELPER_INVOCATION_FEATURES_EXT`

New SPIR-V Capability

- `DemoteToHelperInvocationEXT`

Promotion to Vulkan 1.3

Vulkan APIs in this extension are included in core Vulkan 1.3, with the EXT suffix omitted. External interactions defined by this extension, such as SPIR-V token names, retain their original names. The

original Vulkan API names are still available as aliases of the core functionality.

Version History

- Revision 1, 2019-06-01 (Jeff Bolz)
 - Initial draft

VK_EXT_shader_image_atomic_int64

Name String

VK_EXT_shader_image_atomic_int64

Extension Type

Device extension

Registered Extension Number

235

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

[Vulkan Version 1.1](#)

SPIR-V Dependencies

- [SPV_EXT_shader_image_int64](#)

Contact

- Tobias Hector [\[GitHub\]](#)tobski

Other Extension Metadata

Last Modified Date

2020-07-14

IP Status

No known IP claims.

Interactions and External Dependencies

- This extension provides API support for [GLSL_EXT_shader_image_int64](#)

Contributors

- Matthaeus Chajdas, AMD

- Graham Wihlidal, Epic Games
- Tobias Hector, AMD
- Jeff Bolz, Nvidia
- Faith Ekstrand, Intel

Description

This extension extends existing 64-bit integer atomic support to enable these operations on images as well.

When working with large 2- or 3-dimensional data sets (e.g. rasterization or screen-space effects), image accesses are generally more efficient than equivalent buffer accesses. This extension allows applications relying on 64-bit integer atomics in this manner to quickly improve performance with only relatively minor code changes.

64-bit integer atomic support is guaranteed for optimally tiled images with the `VK_FORMAT_R64_UINT` and `VK_FORMAT_R64_SINT` formats.

New Structures

- Extending `VkPhysicalDeviceFeatures2`, `VkDeviceCreateInfo`:
 - `VkPhysicalDeviceShaderImageAtomicInt64FeaturesEXT`

New Enum Constants

- `VK_EXT_SHADER_IMAGE_ATOMIC_INT64_EXTENSION_NAME`
- `VK_EXT_SHADER_IMAGE_ATOMIC_INT64_SPEC_VERSION`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_IMAGE_ATOMIC_INT64_FEATURES_EXT`

Version History

- Revision 1, 2020-07-14 (Tobias Hector)
 - Initial draft

VK_EXT_shader_stencil_export

Name String

`VK_EXT_shader_stencil_export`

Extension Type

Device extension

Registered Extension Number

141

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

None

SPIR-V Dependencies

- [SPV_EXT_shader_stencil_export](#)

Contact

- Dominik Witczak [\[GitHub\]dominikwiczakamd](#)

Other Extension Metadata

Last Modified Date

2017-07-19

IP Status

No known IP claims.

Interactions and External Dependencies

- This extension provides API support for [GL_ARB_shader_stencil_export](#)

Contributors

- Dominik Witczak, AMD
- Daniel Rakos, AMD
- Rex Xu, AMD

Description

This extension adds support for the SPIR-V extension [SPV_EXT_shader_stencil_export](#), providing a mechanism whereby a shader may generate the stencil reference value per invocation. When stencil testing is enabled, this allows the test to be performed against the value generated in the shader.

New Enum Constants

- [VK_EXT_SHADER_STENCIL_EXPORT_EXTENSION_NAME](#)
- [VK_EXT_SHADER_STENCIL_EXPORT_SPEC_VERSION](#)

Version History

- Revision 1, 2017-07-19 (Dominik Witczak)
 - Initial draft

VK_EXT_subgroup_size_control

Name String

VK_EXT_subgroup_size_control

Extension Type

Device extension

Registered Extension Number

226

Revision

2

Ratification Status

Ratified

Extension and Version Dependencies

[Vulkan Version 1.1](#)

Deprecation State

- *Promoted* to Vulkan 1.3

Contact

- Neil Henning [\[GitHub\]](#)[sheredom](#)

Other Extension Metadata

Last Modified Date

2019-03-05

Contributors

- Jeff Bolz, NVIDIA
- Faith Ekstrand, Intel
- Sławek Grajewski, Intel
- Jesse Hall, Google
- Neil Henning, AMD
- Daniel Koch, NVIDIA
- Jeff Leger, Qualcomm
- Graeme Leese, Broadcom
- Allan MacKinnon, Google
- Mariusz Merecki, Intel
- Graham Wihlidal, Electronic Arts

Description

This extension enables an implementation to control the subgroup size by allowing a varying subgroup size and also specifying a required subgroup size.

It extends the subgroup support in Vulkan 1.1 to allow an implementation to expose a varying subgroup size. Previously Vulkan exposed a single subgroup size per physical device, with the expectation that implementations will behave as if all subgroups have the same size. Some implementations **may** dispatch shaders with a varying subgroup size for different subgroups. As a result they could implicitly split a large subgroup into smaller subgroups or represent a small subgroup as a larger subgroup, some of whose invocations were inactive on launch.

To aid developers in understanding the performance characteristics of their programs, this extension exposes a minimum and maximum subgroup size that a physical device supports and a pipeline create flag to enable that pipeline to vary its subgroup size. If enabled, any **SubgroupSize** decorated variables in the SPIR-V shader modules provided to pipeline creation **may** vary between the **minimum** and **maximum** subgroup sizes.

An implementation is also optionally allowed to support specifying a required subgroup size for a given pipeline stage. Implementations advertise which **stages support a required subgroup size**, and any pipeline of a supported stage can be passed a **VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT** structure to set the subgroup size for that shader stage of the pipeline. For compute shaders, this requires the developer to query the **maxComputeWorkgroupSubgroups** and ensure that:

$$s = \text{WorkGroupSize.x} \times \text{WorkGroupSize.y} \times \text{WorkGroupSize.z} \leq \text{SubgroupSize} \times \text{maxComputeWorkgroupSubgroups}$$

Developers can also specify a new pipeline shader stage create flag that requires the implementation to have fully populated subgroups within local workgroups. This requires the workgroup size in the X dimension to be a multiple of the subgroup size.

New Structures

- Extending **VkPhysicalDeviceFeatures2**, **VkDeviceCreateInfo**:
 - **VkPhysicalDeviceSubgroupSizeControlFeaturesEXT**
- Extending **VkPhysicalDeviceProperties2**:
 - **VkPhysicalDeviceSubgroupSizeControlPropertiesEXT**
- Extending **VkPipelineShaderStageCreateInfo**, **VkShaderCreateInfoEXT**:
 - **VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT**

New Enum Constants

- **VK_EXT_SUBGROUP_SIZE_CONTROL_EXTENSION_NAME**
- **VK_EXT_SUBGROUP_SIZE_CONTROL_SPEC_VERSION**
- Extending **VkPipelineShaderStageCreateFlagBits**:
 - **VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT**
 - **VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT**

- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_FEATURES_EXT](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SUBGROUP_SIZE_CONTROL_PROPERTIES_EXT](#)
 - [VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_REQUIRED_SUBGROUP_SIZE_CREATE_INFO_EXT](#)

Promotion to Vulkan 1.3

Vulkan APIs in this extension are included in core Vulkan 1.3, with the EXT suffix omitted. External interactions defined by this extension, such as SPIR-V token names, retain their original names. The original Vulkan API names are still available as aliases of the core functionality.

Version History

- Revision 1, 2019-03-05 (Neil Henning)
 - Initial draft
- Revision 2, 2019-07-26 (Faith Ekstrand)
 - Add the missing [VkPhysicalDeviceSubgroupSizeControlFeaturesEXT](#) for querying subgroup size control features.

VK_EXT_swapchain_colorspace

Name String

[VK_EXT_swapchain_colorspace](#)

Extension Type

Instance extension

Registered Extension Number

105

Revision

5

Ratification Status

Ratified

Extension and Version Dependencies

[VK_KHR_surface](#)

Contact

- Courtney Goeltzenleuchter [\[GitHub\]](#)[courtney-g](#)

Other Extension Metadata

Last Modified Date

2019-04-26

IP Status

No known IP claims.

Contributors

- Courtney Goeltzenleuchter, Google

Description

This extension expands [VkColorSpaceKHR](#) to add support for most standard color spaces beyond [VK_COLOR_SPACE_SRGB_NONLINEAR_KHR](#). This extension also adds support for [VK_COLOR_SPACE_PASS_THROUGH_EXT](#) which allows applications to use color spaces not explicitly enumerated in [VkColorSpaceKHR](#).

New Enum Constants

- [VK_EXT_SWAPCHAIN_COLOR_SPACE_EXTENSION_NAME](#)
- [VK_EXT_SWAPCHAIN_COLOR_SPACE_SPEC_VERSION](#)
- Extending [VkColorSpaceKHR](#):
 - [VK_COLOR_SPACE_ADOBERGB_LINEAR_EXT](#)
 - [VK_COLOR_SPACE_ADOBERGB_NONLINEAR_EXT](#)
 - [VK_COLOR_SPACE_BT2020_LINEAR_EXT](#)
 - [VK_COLOR_SPACE_BT709_LINEAR_EXT](#)
 - [VK_COLOR_SPACE_BT709_NONLINEAR_EXT](#)
 - [VK_COLOR_SPACE_DCI_P3_NONLINEAR_EXT](#)
 - [VK_COLOR_SPACE_DISPLAY_P3_LINEAR_EXT](#)
 - [VK_COLOR_SPACE_DISPLAY_P3_NONLINEAR_EXT](#)
 - [VK_COLOR_SPACE_DOLBYVISION_EXT](#)
 - [VK_COLOR_SPACE_EXTENDED_SRGB_LINEAR_EXT](#)
 - [VK_COLOR_SPACE_EXTENDED_SRGB_NONLINEAR_EXT](#)
 - [VK_COLOR_SPACE_HDR10_HLG_EXT](#)
 - [VK_COLOR_SPACE_HDR10_ST2084_EXT](#)
 - [VK_COLOR_SPACE_PASS_THROUGH_EXT](#)

Issues

1) Does the spec need to specify which kinds of image formats support the color spaces?

RESOLVED: Pixel format is independent of color space (though some color spaces really want / need floating-point color components to be useful). Therefore, do not plan on documenting what formats support which color spaces. An application **can** call [vkGetPhysicalDeviceSurfaceFormatsKHR](#) to query what a particular implementation supports.

2) How does application determine if HW supports appropriate transfer function for a color space?

RESOLVED: Extension indicates that implementation **must** not do the OETF encoding if it is not sRGB. That responsibility falls to the application shaders. Any other native OETF / EOTF functions supported by an implementation can be described by separate extension.

Version History

- Revision 1, 2016-12-27 (Courtney Goeltzenleuchter)
 - Initial version
- Revision 2, 2017-01-19 (Courtney Goeltzenleuchter)
 - Add pass through and multiple options for BT2020.
 - Clean up some issues with equations not displaying properly.
- Revision 3, 2017-06-23 (Courtney Goeltzenleuchter)
 - Add extended sRGB non-linear enum.
- Revision 4, 2019-04-26 (Graeme Leese)
 - Clarify color space transfer function usage.
 - Refer to normative definitions in the Data Format Specification.
 - Clarify DCI-P3 and Display P3 usage.
- Revision 5, 2024-03-16 (Zehui Lin)
 - Fix interchanged concepts of EOTF and OETF.
 - Clarify that the presentation engine can accept the color spaces.

VK_EXT_texel_buffer_alignment

Name String

VK_EXT_texel_buffer_alignment

Extension Type

Device extension

Registered Extension Number

282

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

Vulkan Version 1.1

Deprecation State

- *Promoted* to Vulkan 1.3

Contact

- Jeff Bolz [[GitHub](#)][jeffbolznv](#)

Other Extension Metadata

Last Modified Date

2019-06-06

IP Status

No known IP claims.

Contributors

- Jeff Bolz, NVIDIA

Description

This extension adds more expressive alignment requirements for uniform and storage texel buffers. Some implementations have single texel alignment requirements that cannot be expressed via [VkPhysicalDeviceLimits::minTexelBufferOffsetAlignment](#).

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT](#)
- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDeviceTexelBufferAlignmentPropertiesEXT](#)

New Enum Constants

- [VK_EXT_TEXEL_BUFFER_ALIGNMENT_EXTENSION_NAME](#)
- [VK_EXT_TEXEL_BUFFER_ALIGNMENT_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_FEATURES_EXT](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXEL_BUFFER_ALIGNMENT_PROPERTIES_EXT](#)

Promotion to Vulkan 1.3

Vulkan APIs in this extension are included in core Vulkan 1.3, with the EXT suffix omitted. However, only the properties structure is promoted. The feature structure is not promoted and [texelBufferAlignment](#) is enabled if using a Vulkan 1.3 instance. External interactions defined by this extension, such as SPIR-V token names, retain their original names. The original Vulkan API name is still available as an alias of the core functionality.

Version History

- Revision 1, 2019-06-06 (Jeff Bolz)
 - Initial draft

VK_EXT_texture_compression_astc_hdr

Name String

VK_EXT_texture_compression_astc_hdr

Extension Type

Device extension

Registered Extension Number

67

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

[Vulkan Version 1.1](#)

Deprecation State

- *Promoted* to Vulkan 1.3

Contact

- Jan-Harald Fredriksen [\[GitHub\]janharaldfredriksen-arm](#)

Other Extension Metadata

Last Modified Date

2019-05-28

IP Status

No known issues.

Contributors

- Jan-Harald Fredriksen, Arm

Description

This extension adds support for textures compressed using the Adaptive Scalable Texture Compression (ASTC) High Dynamic Range (HDR) profile.

When this extension is enabled, the HDR profile is supported for all ASTC formats listed in [ASTC Compressed Image Formats](#).

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceTextureCompressionASTCHDRFeaturesEXT](#)

New Enum Constants

- [VK_EXT_TEXTURE_COMPRESSION_ASTC_HDR_EXTENSION_NAME](#)
- [VK_EXT_TEXTURE_COMPRESSION_ASTC_HDR_SPEC_VERSION](#)
- Extending [VkFormat](#):
 - [VK_FORMAT_ASTC_10x10_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_10x5_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_10x6_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_10x8_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_12x10_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_12x12_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_4x4_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_5x4_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_5x5_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_6x5_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_6x6_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_8x5_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_8x6_SFLOAT_BLOCK_EXT](#)
 - [VK_FORMAT_ASTC_8x8_SFLOAT_BLOCK_EXT](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_TEXTURE_COMPRESSION_ASTC_HDR_FEATURES_EXT](#)

Promotion to Vulkan 1.3

Vulkan APIs in this extension are included in core Vulkan 1.3, with the EXT suffix omitted. However, the feature is made optional in Vulkan 1.3. External interactions defined by this extension, such as SPIR-V token names, retain their original names. The original Vulkan API names are still available as aliases of the core functionality.

Issues

1) Should we add a feature or limit for this functionality?

Yes. It is consistent with the ASTC LDR support to add a feature like `textureCompressionASTC_HDR`.

The feature is strictly speaking redundant as long as this is just an extension; it would be sufficient to just enable the extension. But adding the feature is more forward-looking if wanted to make this an optional core feature in the future.

2) Should we introduce new format enums for HDR?

Yes. Vulkan 1.0 describes the ASTC format enums as UNORM, e.g. `VK_FORMAT_ASTC_4x4_UNORM_BLOCK`, so it is confusing to make these contain HDR data. Note that the OpenGL (ES) extensions did not make this distinction because a single ASTC HDR texture may contain both unorm and float blocks. Implementations **may** not be able to distinguish between LDR and HDR ASTC textures internally and just treat them as the same format, i.e. if this extension is supported then sampling from a `VK_FORMAT_ASTC_4x4_UNORM_BLOCK` image format **may** return HDR results. Applications **can** get predictable results by using the appropriate image format.

Version History

- Revision 1, 2019-05-28 (Jan-Harald Fredriksen)
 - Initial version

VK_EXT_vertex_input_dynamic_state

Name String

`VK_EXT_vertex_input_dynamic_state`

Extension Type

Device extension

Registered Extension Number

353

Revision

2

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_get_physical_device_properties2`

or

[Vulkan Version 1.1](#)

Contact

- Piers Daniell [[GitHub](#)][pdaniell-nv](#)

Other Extension Metadata

Last Modified Date

2020-08-21

IP Status

No known IP claims.

Contributors

- Jeff Bolz, NVIDIA
- Spencer Fricke, Samsung
- Stu Smith, AMD

Description

One of the states that contributes to the combinatorial explosion of pipeline state objects that need to be created, is the vertex input binding and attribute descriptions. By allowing them to be dynamic applications may reduce the number of pipeline objects they need to create.

This extension adds dynamic state support for what is normally static state in [VkPipelineVertexInputStateCreateInfo](#).

New Commands

- [vkCmdSetVertexInputEXT](#)

New Structures

- [VkVertexInputAttributeDescription2EXT](#)
- [VkVertexInputBindingDescription2EXT](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceVertexInputDynamicStateFeaturesEXT](#)

New Enum Constants

- [VK_EXT_VERTEX_INPUT_DYNAMIC_STATE_EXTENSION_NAME](#)
- [VK_EXT_VERTEX_INPUT_DYNAMIC_STATE_SPEC_VERSION](#)
- Extending [VkDynamicState](#):
 - [VK_DYNAMIC_STATE_VERTEX_INPUT_EXT](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VERTEX_INPUT_DYNAMIC_STATE_FEATURES_EXT](#)
 - [VK_STRUCTURE_TYPE_VERTEX_INPUT_ATTRIBUTE_DESCRIPTION_2_EXT](#)
 - [VK_STRUCTURE_TYPE_VERTEX_INPUT_BINDING_DESCRIPTION_2_EXT](#)

Version History

- Revision 2, 2020-11-05 (Piers Daniell)
 - Make [VkVertexInputBindingDescription2EXT](#) extensible
 - Add new [VkVertexInputAttributeDescription2EXT](#) structure for the

`pVertexAttributeDescriptions` parameter to `vkCmdSetVertexInputEXT` so it is also extensible

- Revision 1, 2020-08-21 (Piers Daniell)
 - Internal revisions

VK_EXT_ycbcr_2plane_444_formats

Name String

`VK_EXT_ycbcr_2plane_444_formats`

Extension Type

Device extension

Registered Extension Number

331

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

`VK_KHR_sampler_ycbcr_conversion`

or

[Vulkan Version 1.1](#)

Deprecation State

- *Promoted* to Vulkan 1.3

Contact

- Tony Zlatinski [\[GitHub\]](#)tzlatinski

Other Extension Metadata

Last Modified Date

2020-07-28

IP Status

No known IP claims.

Contributors

- Piers Daniell, NVIDIA
- Ping Liu, Intel

Description

This extension adds some $Y'C_B C_R$ formats that are in common use for video encode and decode, but were not part of the `VK_KHR_sampler_ycbcr_conversion` extension.

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceYcbcr2Plane444FormatsFeaturesEXT](#)

New Enum Constants

- [VK_EXT_YCBCR_2PLANE_444_FORMATS_EXTENSION_NAME](#)
- [VK_EXT_YCBCR_2PLANE_444_FORMATS_SPEC_VERSION](#)
- Extending [VkFormat](#):
 - [VK_FORMAT_G10X6_B10X6R10X6_2PLANE_444_UNORM_3PACK16_EXT](#)
 - [VK_FORMAT_G12X4_B12X4R12X4_2PLANE_444_UNORM_3PACK16_EXT](#)
 - [VK_FORMAT_G16_B16R16_2PLANE_444_UNORM_EXT](#)
 - [VK_FORMAT_G8_B8R8_2PLANE_444_UNORM_EXT](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_2_PLANE_444_FORMATS_FEATURES_EXT](#)

Promotion to Vulkan 1.3

The format enumerants introduced by the extension are included in core Vulkan 1.3, with the EXT suffix omitted. However, runtime support for these formats is optional in core Vulkan 1.3, while if this extension is supported, runtime support is mandatory. The feature structure is not promoted. The original enum names are still available as aliases of the core functionality.

Version History

- Revision 1, 2020-03-08 (Piers Daniell)
 - Initial draft

VK_EXT_ycbcr_image_arrays

Name String

[VK_EXT_ycbcr_image_arrays](#)

Extension Type

Device extension

Registered Extension Number

253

Revision

1

Ratification Status

Ratified

Extension and Version Dependencies

VK_KHR_sampler_ycbcr_conversion

or

Vulkan Version 1.1

Contact

- Piers Daniell [[GitHub](#)][pdaniell-nv](#)

Other Extension Metadata

Last Modified Date

2019-01-15

Contributors

- Piers Daniell, NVIDIA

Description

This extension allows images of a format that requires [Y'C_BC_R conversion](#) to be created with multiple array layers, which is otherwise restricted.

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceYcbcrImageArraysFeaturesEXT](#)

New Enum Constants

- [VK_EXT_YCBCR_IMAGE_ARRAYS_EXTENSION_NAME](#)
- [VK_EXT_YCBCR_IMAGE_ARRAYS_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_YCBCR_IMAGE_ARRAYS_FEATURES_EXT](#)

Version History

- Revision 1, 2019-01-15 (Piers Daniell)
 - Initial revision

VK_NV_acquire_winrt_display

Name String

[VK_NV_acquire_winrt_display](#)

Extension Type

Device extension

Registered Extension Number

346

Revision

1

Ratification Status

Not ratified

Extension and Version Dependencies

[VK_EXT_direct_mode_display](#)

Contact

- Jeff Juliano [\[GitHub\]](#)[jjuliano](#)

Other Extension Metadata

Last Modified Date

2020-09-29

IP Status

No known IP claims.

Contributors

- Jeff Juliano, NVIDIA

Description

This extension allows an application to take exclusive control of a display on Windows 10 provided that the display is not already controlled by a compositor. Examples of compositors include the Windows desktop compositor, other applications using this Vulkan extension, and applications that “[Acquire](#)” a “[DisplayTarget](#)” using a “[WinRT](#)” command such as “[winrt::Windows::Devices::Display::Core::DisplayManager.TryAcquireTarget\(\)](#)”.

When control is acquired the application has exclusive access to the display until control is released or the application terminates. An application’s attempt to acquire is denied if a different application has already acquired the display.

New Commands

- [vkAcquireWinrtDisplayNV](#)
- [vkGetWinrtDisplayNV](#)

New Enum Constants

- [VK_NV_ACQUIRE_WINRT_DISPLAY_EXTENSION_NAME](#)
- [VK_NV_ACQUIRE_WINRT_DISPLAY_SPEC_VERSION](#)

Issues

1) What should the platform substring be for this extension:

RESOLVED: The platform substring is “Winrt”.

The substring “Winrt” matches the fact that the OS API exposing the acquire and release functionality is called “WinRT”.

The substring “Win32” is wrong because the related “WinRT” API is explicitly **not** a “Win32” API. “WinRT” is a competing API family to the “Win32” API family.

The substring “Windows” is suboptimal because there could be more than one relevant API on the Windows platform. There is preference to use the more-specific substring “Winrt”.

2) Should [vkAcquireWinrtDisplayNV](#) take a winRT DisplayTarget, or a Vulkan display handle as input?

RESOLVED: A Vulkan display handle. This matches the design of [vkAcquireXlibDisplayEXT](#).

3) Should the acquire command be platform-independent named “vkAcquireDisplayNV”, or platform-specific named “vkAcquireWinrtDisplayNV”?

RESOLVED: Add a platform-specific command.

The inputs to the Acquire command are all Vulkan types. None are WinRT types. This opens the possibility of the winrt extension defining a platform-independent acquire command.

The X11 acquire command does need to accept a platform-specific parameter. This could be handled by adding to a platform-independent acquire command a params structure to which platform-dependent types can be chained by [pNext](#) pointer.

The prevailing opinion is that it would be odd to create a second platform-independent function that is used on the Windows 10 platform, but that is not used for the X11 platform. Since a Windows 10 platform-specific command is needed anyway for converting between vkDisplayKHR and platform-native handles, opinion was to create a platform-specific acquire function.

4) Should the [vkGetWinrtDisplayNV](#) parameter identifying a display be named “deviceRelativeId” or “adapterRelativeId”?

RESOLVED: The WinRT name is “AdapterRelativeId”. The name “adapter” is the Windows analog to a Vulkan “physical device”. Vulkan already has precedent to use the name [deviceLUID](#) for the concept that Windows APIs call “AdapterLuid”. Keeping form with this precedent, the name “deviceRelativeId” is chosen.

5) Does [vkAcquireWinrtDisplayNV](#) cause the Windows desktop compositor to release a display?

RESOLVED: No. [vkAcquireWinrtDisplayNV](#) does not itself cause the Windows desktop compositor to release a display. This action must be performed outside of Vulkan.

Beginning with Windows 10 version 2004 it is possible to cause the Windows desktop compositor to release a display by using the “Advanced display settings” sub-page of the “Display settings” control

panel. See <https://docs.microsoft.com/en-us/windows-hardware/drivers/display/specialized-monitors>

6) Where can one find additional information about custom compositors for Windows 10?

RESOLVED: Relevant references are as follows.

According to Microsoft's documentation on "[building a custom compositor](#)", the ability to write a custom compositor is not a replacement for a fullscreen desktop window. The feature is for writing compositor apps that drive specialized hardware.

Only certain editions of Windows 10 support custom compositors, "[documented here](#)". The product type can be queried from Windows 10. See <https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-getproductinfo>

Version History

- Revision 1, 2020-09-29 (Jeff Juliano)
 - Initial draft

VK_NV_external_memory_sci_buf

Name String

VK_NV_external_memory_sci_buf

Extension Type

Device extension

Registered Extension Number

375

Revision

2

Ratification Status

Not ratified

Extension and Version Dependencies

[Vulkan Version 1.1](#)

Contact

- Kai Zhang [[GitHub](#)][kazhang](#)

Other Extension Metadata

Last Modified Date

2022-04-12

Contributors

- Kai Zhang, NVIDIA
- Jeff Bolz, NVIDIA
- Jonathan McCaffrey, NVIDIA
- Daniel Koch, NVIDIA

Description

This extension enables an application to access external memory via [NvSciBufObj](#). To import a [NvSciBufObj](#) to [VkDeviceMemory](#), applications need to:

- Create an unreconciled [NvSciBufAttrList](#) via [NvSciBufAttrListCreate\(\)](#)
- Fill in the private attribute list via [vkGetPhysicalDeviceSciBufAttributesNV\(\)](#)
- Fill in the public attribute list via [NvSciBufAttrListSetAttrs\(\)](#)
- Reconcile the [NvSciBufAttrList](#) via [NvSciBufAttrListReconcile\(\)](#)
- Create a [NvSciBufObj](#) via [NvSciBufObjAlloc\(\)](#)
- Import the [NvSciBufObj](#) to a [VkDeviceMemory](#) by chaining [VkImportMemorySciBufInfoNV](#) structure to the command [vkAllocateMemory](#).

For details of the [NvSciBuf](#) APIs and data structures, see the [NvStreams Documentation](#).

New Commands

- [vkGetMemorySciBufNV](#)
- [vkGetPhysicalDeviceExternalMemorySciBufPropertiesNV](#)
- [vkGetPhysicalDeviceSciBufAttributesNV](#)

New Structures

- [VkMemoryGetSciBufInfoNV](#)
- [VkMemorySciBufPropertiesNV](#)
- Extending [VkMemoryAllocateInfo](#):
 - [VkExportMemorySciBufInfoNV](#)
 - [VkImportMemorySciBufInfoNV](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceExternalMemorySciBufFeaturesNV](#)
 - [VkPhysicalDeviceExternalSciBufFeaturesNV](#)

New Enum Constants

- [VK_NV_EXTERNAL_MEMORY_SCI_BUF_EXTENSION_NAME](#)
- [VK_NV_EXTERNAL_MEMORY_SCI_BUF_SPEC_VERSION](#)

- Extending [VkExternalMemoryHandleTypeFlagBits](#):
 - `VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCI_BUF_BIT_NV`
- Extending [VkStructureType](#):
 - `VK_STRUCTURE_TYPE_EXPORT_MEMORY_SCI_BUF_INFO_NV`
 - `VK_STRUCTURE_TYPE_IMPORT_MEMORY_SCI_BUF_INFO_NV`
 - `VK_STRUCTURE_TYPE_MEMORY_GET_SCI_BUF_INFO_NV`
 - `VK_STRUCTURE_TYPE_MEMORY_SCI_BUF_PROPERTIES_NV`
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_SCI_BUF_FEATURES_NV`
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SCI_BUF_FEATURES_NV`

Issues

1) What should we call this extension?

RESOLVED. The external API is `NvSciBuf`, but the Vulkan convention is to append the vendor suffix at the end of an identifier. Using `NvSciBufNV` seems awkward, so we have chosen to use just the `SciBuf` portion of the name in Vulkan commands and tokens. Since this is for interacting with memory objects allocated from outside Vulkan, we use "external_memory" in the name, similar to [VK_KHR_external_memory_fd](#). To avoid an explosion of extensions, we include the capability to import and export memory in one extension but include separate features in case implementations only implement (or safety certify) a subset.

2) What changed in revision 2?

RESOLVED. The [VkPhysicalDeviceExternalSciBufFeaturesNV](#) structure was renamed to [VkPhysicalDeviceExternalMemorySciBufFeaturesNV](#) to follow naming conventions (previous names retained as aliases), and drop const on `pNext` pointer.

Version History

- Revision 1, 2022-04-12 (Kai Zhang, Daniel Koch)
 - Internal revisions
- Revision 2, 2023-01-03 (Daniel Koch)
 - fix the feature structure to address naming convention and cts autogeneration issues

VK_NV_external_sci_sync2

Name String

`VK_NV_external_sci_sync2`

Extension Type

Device extension

Registered Extension Number

490

Revision

1

Ratification Status

Not ratified

Extension and Version Dependencies

[Vulkan Version 1.1](#)

API Interactions

- Interacts with `VKSC_VERSION_1_0`

Contact

- Kai Zhang [[GitHub](#)][kazhang](#)

Other Extension Metadata

Last Modified Date

2022-09-07

Contributors

- Kai Zhang, NVIDIA
- Jeff Bolz, NVIDIA
- Jonathan McCaffrey, NVIDIA
- Daniel Koch, NVIDIA

Description

An application using external memory may wish to synchronize access to that memory using semaphores and fences. This extension enables an application to import semaphore and import/export fence payloads to and from `NvSciSync` objects. To import a `NvSciSyncObj` to a `VkSemaphore` or `VkFence`, applications need to:

- Create an unreconciled `NvSciSyncAttrList` via `NvSciSyncAttrListCreate()`
- Fill the private attribute list via `vkGetPhysicalDeviceSciSyncAttributesNV()`
- Fill the public attribute list via `NvSciSyncAttrListSetAttrs()`
- Reconcile the `NvSciSyncAttrList` via `NvSciSyncAttrListReconcile()`
- Create a `NvSciSyncObj` via `NvSciSyncObjAlloc()`
- To import a `NvSciSyncObj` to a `VkSemaphore`, create a `VkSemaphoreSciSyncPoolNV` for the `NvSciSyncObj` and then select the semaphore from `VkSemaphoreSciSyncPoolNV` by passing the `VkSemaphoreSciSyncCreateInfoNV` structure to `vkCreateSemaphore`
- To import a `NvSciSyncObj` to a `VkFence`, pass the `VkImportFenceSciSyncInfoNV` structure to the `vkImportFenceSciSyncObjNV` command.

To import/export a `NvSciSyncFence` to a `VkFence` object, that `VkFence` object **must** already have a `NvSciSyncObj` previously imported.

This extension does not support exporting semaphores from **NvSciSync** objects.

For details of the **NvSciSync** APIs and data structures, see the **NvStreams Documentation**.

New Object Types

- [VkSemaphoreSciSyncPoolNV](#)

New Commands

- [vkCreateSemaphoreSciSyncPoolNV](#)
- [vkDestroySemaphoreSciSyncPoolNV](#)
- [vkGetFenceSciSyncFenceNV](#)
- [vkGetFenceSciSyncObjNV](#)
- [vkGetPhysicalDeviceSciSyncAttributesNV](#)
- [vkImportFenceSciSyncFenceNV](#)
- [vkImportFenceSciSyncObjNV](#)

New Structures

- [VkFenceGetSciSyncInfoNV](#)
- [VkImportFenceSciSyncInfoNV](#)
- [VkSciSyncAttributesInfoNV](#)
- [VkSemaphoreSciSyncPoolCreateInfoNV](#)
- Extending [VkFenceCreateInfo](#):
 - [VkExportFenceSciSyncInfoNV](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceExternalSciSync2FeaturesNV](#)
- Extending [VkSemaphoreCreateInfo](#):
 - [VkSemaphoreSciSyncCreateInfoNV](#)

If Vulkan SC 1.0 is supported:

- Extending [VkDeviceCreateInfo](#):
 - [VkDeviceSemaphoreSciSyncPoolReservationCreateInfoNV](#)

New Enums

- [VkSciSyncClientTypeNV](#)
- [VkSciSyncPrimitiveTypeNV](#)

New Enum Constants

- `VK_NV_EXTERNAL_SCI_SYNC_2_EXTENSION_NAME`
- `VK_NV_EXTERNAL_SCI_SYNC_2_SPEC_VERSION`
- Extending `VkExternalFenceHandleTypeFlagBits`:
 - `VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV`
 - `VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV`
- Extending `VkObjectType`:
 - `VK_OBJECT_TYPE_SEMAPHORE_SCI_SYNC_POOL_NV`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_EXPORT_FENCE_SCI_SYNC_INFO_NV`
 - `VK_STRUCTURE_TYPE_FENCE_GET_SCI_SYNC_INFO_NV`
 - `VK_STRUCTURE_TYPE_IMPORT_FENCE_SCI_SYNC_INFO_NV`
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SCI_SYNC_2_FEATURES_NV`
 - `VK_STRUCTURE_TYPE_SCI_SYNC_ATTRIBUTES_INFO_NV`
 - `VK_STRUCTURE_TYPE_SEMAPHORE_SCI_SYNC_CREATE_INFO_NV`
 - `VK_STRUCTURE_TYPE_SEMAPHORE_SCI_SYNC_POOL_CREATE_INFO_NV`

If Vulkan SC 1.0 is supported:

- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_DEVICE_SEMAPHORE_SCI_SYNC_POOL_RESERVATION_CREATE_INFO_NV`

Issues

1) Does this extension extend or replace `VK_NV_external_sci_sync`?

RESOLVED. Replaces - expect to deprecate it and eventually remove it.

2) What part of `VK_NV_external_sci_sync` is deprecated/removed in this extension?

RESOLVED. The commands to import and export semaphores from `VK_NV_external_sci_sync` are removed and have been replaced with an alternate mechanism to import semaphores. Fence import and export functionality is unchanged.

In particular:

- Removed Commands:
 - `vkImportSemaphoreSciSyncObjNV`
 - `vkGetSemaphoreSciSyncObjNV`
- Removed Structures:
 - `VkImportSemaphoreSciSyncInfoNV`

- [VkExportSemaphoreSciSyncInfoNV](#)
- [VkSemaphoreGetSciSyncInfoNV](#)

3) Application migration guide from [VK_NV_external_sci_sync](#) to [VK_NV_external_sci_sync2](#)

- In [VK_NV_external_sci_sync](#), to import a [NvSciSyncObj](#) to [VkSemaphore](#), applications need to:
 - Create a [VkSemaphore](#) by command [vkCreateSemaphore](#).
 - Call [vkImportSemaphoreSciSyncObjNV](#) command to import the [NvSciSyncObj](#) to [VkSemaphore](#) created.
 - Call [vkDestroySemaphore](#) to destroy the [VkSemaphore](#) after all submitted batches that refer to it have completed execution.
- In order to migrate to [VK_NV_external_sci_sync2](#), applications need to:
 - Chain [VkDeviceSemaphoreSciSyncPoolReservationCreateInfoNV](#) to [VkDeviceObjectReservationCreateInfo](#) and specify the [semaphoreSciSyncPoolRequestCount](#) maximum number of semaphore SciSync pools that will be used simultaneously.
 - Import the a [NvSciSyncObj](#) to a [VkSemaphoreSciSyncPoolNV](#) by command [vkCreateSemaphoreSciSyncPoolNV](#).
 - Select the [VkSemaphore](#) from [VkSemaphoreSciSyncPoolNV](#) by passing the [VkSemaphoreSciSyncCreateInfoNV](#) structure to [vkCreateSemaphore](#).
 - Can call [vkDestroySemaphore](#) to destroy the [VkSemaphore](#) immediately after all the batches that refer to it are submitted.

Version History

- Revision 1, 2022-09-07 (Kai Zhang, Daniel Koch)
 - Initial revision

VK_NV_private_vendor_info

Name String

[VK_NV_private_vendor_info](#)

Extension Type

Device extension

Registered Extension Number

52

Revision

2

Ratification Status

Not ratified

Extension and Version Dependencies

None

Contact

- Daniel Koch [\[GitHub\]dgkoch](#)

Other Extension Metadata

Last Modified Date

2022-08-10

Contributors

- Daniel Koch, NVIDIA
- Jonathan McCaffrey, NVIDIA
- Jeff Bolz, NVIDIA

Description

This extension provides the application with access to vendor-specific enums and structures that are not expected to be publicly documented.

New Enum Constants

- `VK_NV_PRIVATE_VENDOR_INFO_EXTENSION_NAME`
- `VK_NV_PRIVATE_VENDOR_INFO_SPEC_VERSION`
- Extending [VkStructureType](#):
 - `VK_STRUCTURE_TYPE_PRIVATE_VENDOR_INFO_PLACEHOLDER_OFFSET_0_NV`

Issues

1) What should we call this extension?

RESOLVED. [VK_NV_private_vendor_info](#) as this contains details of NVIDIA's implementation that we do not expect to publicly document.

Version History

- Revision 1, 2022-05-03 (Daniel Koch)
 - Internal revisions
- Revision 2, 2022-08-10 (Daniel Koch)
 - change number for extension (373 to 52) to avoid conflict

VK_QNX_external_memory_screen_buffer

Name String

`VK_QNX_external_memory_screen_buffer`

Extension Type

Device extension

Registered Extension Number

530

Revision

1

Ratification Status

Not ratified

Extension and Version Dependencies

`VK_KHR_sampler_ycbcr_conversion`
and
`VK_KHR_external_memory`
and
`VK_KHR_dedicated_allocation`
or
[Vulkan Version 1.1](#)
and
[VK_EXT_queue_family_foreign](#)

Contact

- Mike Gorchak [\[GitHub\]mgorchak-blackberry](#)
- Aaron Ruby [\[GitHub\]aruby-blackberry](#)

Other Extension Metadata

Last Modified Date

2023-05-17

IP Status

No known IP claims.

Contributors

- Mike Gorchak, QNX / Blackberry Limited
- Aaron Ruby, QNX / Blackberry Limited

Description

This extension enables an application to import QNX Screen `_screen_buffer` objects created outside of the Vulkan device into Vulkan memory objects, where they can be bound to images and buffers.

Some `_screen_buffer` images have implementation-defined *external formats* that **may** not correspond to Vulkan formats. Sampler Y'C_BC_R conversion **can** be used to sample from these images and convert them to a known color space.

`_screen_buffer` is strongly typed, so naming the handle type is redundant. The internal layout and therefore size of a `_screen_buffer` image may depend on native usage flags that do not have corresponding Vulkan counterparts.

New Commands

- [vkGetScreenBufferPropertiesQNX](#)

New Structures

- [VkScreenBufferPropertiesQNX](#)
- Extending [VkImageCreateInfo](#), [VkSamplerYcbcrConversionCreateInfo](#):
 - [VkExternalFormatQNX](#)
- Extending [VkMemoryAllocateInfo](#):
 - [VkImportScreenBufferInfoQNX](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceExternalMemoryScreenBufferFeaturesQNX](#)
- Extending [VkScreenBufferPropertiesQNX](#):
 - [VkScreenBufferFormatPropertiesQNX](#)

New Enum Constants

- [VK_QNX_EXTERNAL_MEMORY_SCREEN_BUFFER_EXTENSION_NAME](#)
- [VK_QNX_EXTERNAL_MEMORY_SCREEN_BUFFER_SPEC_VERSION](#)
- Extending [VkExternalMemoryHandleTypeFlagBits](#):
 - [VK_EXTERNAL_MEMORY_HANDLE_TYPE_SCREEN_BUFFER_BIT_QNX](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_EXTERNAL_FORMAT_QNX](#)
 - [VK_STRUCTURE_TYPE_IMPORT_SCREEN_BUFFER_INFO_QNX](#)
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_MEMORY_SCREEN_BUFFER_FEATURES_QNX](#)
 - [VK_STRUCTURE_TYPE_SCREEN_BUFFER_FORMAT_PROPERTIES_QNX](#)
 - [VK_STRUCTURE_TYPE_SCREEN_BUFFER_PROPERTIES_QNX](#)

Issues

Version History

- Revision 1, 2023-05-17 (Mike Gorchak)
 - Initial version

List of Deprecated Extensions

- [VK_EXT_index_type_uint8](#)
- [VK_EXT_line_rasterization](#)
- [VK_EXT_validation_features](#)
- [VK_NV_external_sci_sync](#)

VK_EXT_index_type_uint8

Name String

VK_EXT_index_type_uint8

Extension Type

Device extension

Registered Extension Number

266

Revision

1

Ratification Status

Not ratified

Extension and Version Dependencies

VK_KHR_get_physical_device_properties2

or

[Vulkan Version 1.1](#)

Deprecation State

- *Promoted* to [VK_KHR_index_type_uint8](#) extension
 - Which in turn was *promoted* to Vulkan 1.4

Contact

- Piers Daniell [\[GitHub\]](#)[pdaniell-nv](#)

Other Extension Metadata

Last Modified Date

2019-05-02

IP Status

No known IP claims.

Contributors

- Jeff Bolz, NVIDIA

Description

This extension allows `uint8_t` indices to be used with [vkCmdBindIndexBuffer](#).

Promotion to VK_KHR_index_type_uint8

All functionality in this extension is included in [VK_KHR_index_type_uint8](#), with the suffix changed to KHR. The original enum names are still available as aliases of the KHR functionality.

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceIndexTypeUint8FeaturesEXT](#)

New Enum Constants

- [VK_EXT_INDEX_TYPE_UINT8_EXTENSION_NAME](#)
- [VK_EXT_INDEX_TYPE_UINT8_SPEC_VERSION](#)
- Extending [VkIndexType](#):
 - [VK_INDEX_TYPE_UINT8_EXT](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_INDEX_TYPE_UINT8_FEATURES_EXT](#)

Version History

- Revision 1, 2019-05-02 (Piers Daniell)
 - Internal revisions

VK_EXT_line_rasterization

Name String

[VK_EXT_line_rasterization](#)

Extension Type

Device extension

Registered Extension Number

260

Revision

1

Ratification Status

Not ratified

Extension and Version Dependencies

[VK_KHR_get_physical_device_properties2](#)

or

[Vulkan Version 1.1](#)

Deprecation State

- *Promoted* to [VK_KHR_line_rasterization](#) extension
 - Which in turn was *promoted* to Vulkan 1.4

Special Use

- [CAD support](#)

Contact

- Jeff Bolz [\[GitHub\]jeffbolz](#)

Other Extension Metadata

Last Modified Date

2019-05-09

IP Status

No known IP claims.

Contributors

- Jeff Bolz, NVIDIA
- Allen Jensen, NVIDIA
- Faith Ekstrand, Intel

Description

This extension adds some line rasterization features that are commonly used in CAD applications and supported in other APIs like OpenGL. Bresenham-style line rasterization is supported, smooth rectangular lines (coverage to alpha) are supported, and stippled lines are supported for all three line rasterization modes.

Promotion to **VK_KHR_line_rasterization**

All functionality in this extension is included in [VK_KHR_line_rasterization](#), with the suffix changed to KHR. The original enum names are still available as aliases of the KHR functionality.

New Commands

- [vkCmdSetLineStippleEXT](#)

New Structures

- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceLineRasterizationFeaturesEXT](#)
- Extending [VkPhysicalDeviceProperties2](#):
 - [VkPhysicalDeviceLineRasterizationPropertiesEXT](#)
- Extending [VkPipelineRasterizationStateCreateInfo](#):
 - [VkPipelineRasterizationLineStateCreateInfoEXT](#)

New Enums

- [VkLineRasterizationModeEXT](#)

New Enum Constants

- `VK_EXT_LINE_RASTERIZATION_EXTENSION_NAME`
- `VK_EXT_LINE_RASTERIZATION_SPEC_VERSION`
- Extending `VkDynamicState`:
 - `VK_DYNAMIC_STATE_LINE_STIPPLE_EXT`
- Extending `VkLineRasterizationMode`:
 - `VK_LINE_RASTERIZATION_MODE_BRESENHAM_EXT`
 - `VK_LINE_RASTERIZATION_MODE_DEFAULT_EXT`
 - `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_EXT`
 - `VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_EXT`
- Extending `VkStructureType`:
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_FEATURES_EXT`
 - `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_LINE_RASTERIZATION_PROPERTIES_EXT`
 - `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_LINE_STATE_CREATE_INFO_EXT`

Issues

1) Do we need to support Bresenham-style and smooth lines with more than one rasterization sample? i.e. the equivalent of `glDisable(GL_MULTISAMPLE)` in OpenGL when the framebuffer has more than one sample?

RESOLVED: Yes. For simplicity, Bresenham line rasterization carries forward a few restrictions from OpenGL, such as not supporting per-sample shading, alpha to coverage, or alpha to one.

Version History

- Revision 1, 2019-05-09 (Jeff Bolz)
 - Initial draft

VK_EXT_validation_features

Name String

`VK_EXT_validation_features`

Extension Type

Instance extension

Registered Extension Number

248

Revision

6

Ratification Status

Not ratified

Extension and Version Dependencies

None

Deprecation State

- *Deprecated* by [VK_EXT_layer_settings](#) extension

Special Use

- [Debugging tools](#)

Contact

- Karl Schultz [\[GitHub\]](#)[karl-lunarg](#)

Other Extension Metadata

Last Modified Date

2018-11-14

IP Status

No known IP claims.

Contributors

- Karl Schultz, LunarG
- Dave Houlton, LunarG
- Mark Lobodzinski, LunarG
- Camden Stocker, LunarG
- Tony Barbour, LunarG
- John Zulauf, LunarG

Description

This extension provides the [VkValidationFeaturesEXT](#) structure that can be included in the [pNext](#) chain of the [VkInstanceCreateInfo](#) structure passed as the [pCreateInfo](#) parameter of [vkCreateInstance](#). The structure contains an array of [VkValidationFeatureEnableEXT](#) enum values that enable specific validation features that are disabled by default. The structure also contains an array of [VkValidationFeatureDisableEXT](#) enum values that disable specific validation layer features that are enabled by default.

Deprecation by [VK_EXT_layer_settings](#)

Functionality in this extension is subsumed into the [VK_EXT_layer_settings](#) extension.

New Structures

- Extending [VkInstanceCreateInfo](#), [VkShaderModuleCreateInfo](#), [VkShaderCreateInfoEXT](#):

- [VkValidationFeaturesEXT](#)

New Enums

- [VkValidationFeatureDisableEXT](#)
- [VkValidationFeatureEnableEXT](#)

New Enum Constants

- [VK_EXT_VALIDATION_FEATURES_EXTENSION_NAME](#)
- [VK_EXT_VALIDATION_FEATURES_SPEC_VERSION](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_VALIDATION_FEATURES_EXT](#)

Version History

- Revision 1, 2018-11-14 (Karl Schultz)
 - Initial revision
- Revision 2, 2019-08-06 (Mark Lobodzinski)
 - Add Best Practices enable
- Revision 3, 2020-03-04 (Tony Barbour)
 - Add Debug Printf enable
- Revision 4, 2020-07-29 (John Zulauf)
 - Add Synchronization Validation enable
- Revision 5, 2021-05-18 (Tony Barbour)
 - Add Shader Validation Cache disable
- Revision 6, 2023-09-25 (Christophe Riccio)
 - Marked as deprecated by [VK_EXT_layer_settings](#)

VK_NV_external_sci_sync

Name String

[VK_NV_external_sci_sync](#)

Extension Type

Device extension

Registered Extension Number

374

Revision

2

Ratification Status

Not ratified

Extension and Version Dependencies

[Vulkan Version 1.1](#)

Deprecation State

- *Deprecated* by [VK_NV_external_sci_sync2](#) extension

Contact

- Kai Zhang [[GitHub](#)][kazhang](#)

Other Extension Metadata

Last Modified Date

2022-04-12

Contributors

- Kai Zhang, NVIDIA
- Jeff Bolz, NVIDIA
- Jonathan McCaffrey, NVIDIA
- Daniel Koch, NVIDIA

Description

An application using external memory may wish to synchronize access to that memory using semaphores and fences. This extension enables an application to import and export semaphore and fence payloads to and from [NvSciSync](#) objects. To import a [NvSciSyncObj](#) to a [VkSemaphore](#) or [VkFence](#), applications need to:

- Create an unreconciled [NvSciSyncAttrList](#) via [NvSciSyncAttrListCreate\(\)](#)
- Fill the private attribute list via [vkGetPhysicalDeviceSciSyncAttributesNV\(\)](#)
- Fill the public attribute list via [NvSciSyncAttrListSetAttrs\(\)](#)
- Reconcile the [NvSciSyncAttrList](#) via [NvSciSyncAttrListReconcile\(\)](#)
- Create a [NvSciSyncObj](#) via [NvSciSyncObjAlloc\(\)](#)
- Import the [NvSciSyncObj](#) to a [VkSemaphore](#) by passing the [VkImportSemaphoreSciSyncInfoNV](#) structure to the [vkImportSemaphoreSciSyncObjNV](#) command, or to a [VkFence](#) by passing the [VkImportFenceSciSyncInfoNV](#) structure to the [vkImportFenceSciSyncObjNV](#) command.

To import/export a [NvSciSyncFence](#) to a [VkFence](#) object, that [VkFence](#) object **must** already have a [NvSciSyncObj](#) previously imported.

For details of the [NvSciSync](#) APIs and data structures, see the [NvStreams Documentation](#).

New Commands

- [vkGetFenceSciSyncFenceNV](#)
- [vkGetFenceSciSyncObjNV](#)
- [vkGetPhysicalDeviceSciSyncAttributesNV](#)
- [vkGetSemaphoreSciSyncObjNV](#)
- [vkImportFenceSciSyncFenceNV](#)
- [vkImportFenceSciSyncObjNV](#)
- [vkImportSemaphoreSciSyncObjNV](#)

New Structures

- [VkFenceGetSciSyncInfoNV](#)
- [VkImportFenceSciSyncInfoNV](#)
- [VkImportSemaphoreSciSyncInfoNV](#)
- [VkSciSyncAttributesInfoNV](#)
- [VkSemaphoreGetSciSyncInfoNV](#)
- Extending [VkFenceCreateInfo](#):
 - [VkExportFenceSciSyncInfoNV](#)
- Extending [VkPhysicalDeviceFeatures2](#), [VkDeviceCreateInfo](#):
 - [VkPhysicalDeviceExternalSciSyncFeaturesNV](#)
- Extending [VkSemaphoreCreateInfo](#):
 - [VkExportSemaphoreSciSyncInfoNV](#)

New Enums

- [VkSciSyncClientTypeNV](#)
- [VkSciSyncPrimitiveTypeNV](#)

New Enum Constants

- [VK_NV_EXTERNAL_SCI_SYNC_EXTENSION_NAME](#)
- [VK_NV_EXTERNAL_SCI_SYNC_SPEC_VERSION](#)
- Extending [VkExternalFenceHandleTypeFlagBits](#):
 - [VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_FENCE_BIT_NV](#)
 - [VK_EXTERNAL_FENCE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV](#)
- Extending [VkExternalSemaphoreHandleTypeFlagBits](#):
 - [VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_SCI_SYNC_OBJ_BIT_NV](#)
- Extending [VkStructureType](#):
 - [VK_STRUCTURE_TYPE_EXPORT_FENCE_SCI_SYNC_INFO_NV](#)

- `VK_STRUCTURE_TYPE_EXPORT_SEMAPHORE_SCI_SYNC_INFO_NV`
- `VK_STRUCTURE_TYPE_FENCE_GET_SCI_SYNC_INFO_NV`
- `VK_STRUCTURE_TYPE_IMPORT_FENCE_SCI_SYNC_INFO_NV`
- `VK_STRUCTURE_TYPE_IMPORT_SEMAPHORE_SCI_SYNC_INFO_NV`
- `VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_EXTERNAL_SCI_SYNC_FEATURES_NV`
- `VK_STRUCTURE_TYPE_SCI_SYNC_ATTRIBUTES_INFO_NV`
- `VK_STRUCTURE_TYPE_SEMAPHORE_GET_SCI_SYNC_INFO_NV`

Issues

1) What should we call this extension?

RESOLVED. The external API is `NvSciSync`, but the Vulkan convention is to append the vendor suffix at the end of an identifier. Using `NvSciSyncNV` seems awkward, so we have chosen to use just the `SciSync` portion of the name in Vulkan commands and tokens. Since this is for interacting with objects from outside Vulkan, we use "external" in the name, similar to `VK_KHR_external_fence_fd`. To avoid an explosion of extensions, we include the capability to import and export both semaphores and fences in one extension but include separate features in case implementations only implement (or safety certify) a subset.

2) How do we resolve the NvStreams terminology of `NvSciSyncFence` which conflicts with the Vulkan SC terminology of `VkFence`.

RESOLVED: "fence" refers to `VkFence`. "`NvSciSyncFence`" refers to the NvStreams type and "`VkFence`" refers to the Vulkan SC type.

Version History

- Revision 2, 2022-03-29 (Daniel Koch)
 - use separate commands for `NvSciSyncFence` and `NvSciSyncObj` handles
- Revision 1, 2020-11-25 (Kai Zhang, Daniel Koch)
 - Initial revision

Appendix F: Legacy and Superseded Functionality

Functionality in the specification such as commands and structures **may** be marked as *legacy*. This label indicates that either functionality that supersedes it is available in a newer core version, or the functionality has simply fallen out of favor for other reasons. Forward-looking applications **should** avoid using functionality marked as legacy where possible.

A link to an explanatory section is provided to explain why the functionality was marked as legacy, what functionality supersedes it (if any), and what applications **should** do instead.

Newer extensions **may** also provide functionality that supersedes existing functionality, but the original functionality will not be tagged as legacy in the specification unless a superseding solution has been fully promoted to the core API. Explanatory text will still be included below for these cases as reference.

NOTE

Legacy and superseded functionality is tagged in the xml registry via a deprecation tag, for both core versions and extensions. It is expected that tooling such as the validation layers will be able to warn users when they are making use of any functionality that is either legacy or superseded by the version and extensions the application has enabled.

Legacy functionality remains available for use in the specification, but it **may** be less capable than newer functionality. Interactions with legacy functionality will often be omitted when new extensions or features are developed, so legacy functionality **may** not work with the latest features.

NOTE

Another mechanism exists to indicate that an entire extension is considered legacy, using different terminology. See [the Deprecation section of the Extensions chapter](#) for more information.

List of Legacy Functionality

When functionality is marked as legacy or if an extension supersedes it, an explanation will be added to the following sections.

NOTE

This list is a work in progress - we intend to add other items to this section over time. Examples include shader modules (superseded by [VK_KHR_maintenance5](#)) and anything superseded by a new version of the same functionality (e.g. [VK_KHR_synchronization2](#)).

Physical Device Queries: Superseded via version 2

[VK_KHR_get_physical_device_properties2](#) was incorporated into Vulkan 1.1, which introduced new versions of several physical device query functions. These provide the same functionality as the Vulkan 1.0 functionality but with greater extensibility.

When querying device features, `vkGetPhysicalDeviceFeatures2` **should** be used instead of `vkGetPhysicalDeviceFeatures`. When enabling device features, `VkPhysicalDeviceFeatures2` **should** be provided in the `pNext` chain of `VkDeviceCreateInfo` instead of using `VkDeviceCreateInfo::pEnabledFeatures`.

Device Layers: Superseded via instance layers

Previous versions of this specification distinguished between instance and device layers. Instance layers were only able to intercept commands that operate on `VkInstance` and `VkPhysicalDevice`, except they were not able to intercept `vkCreateDevice`. Device layers were enabled for individual devices when they were created, and could only intercept commands operating on that device or its child objects.

Device-only layers are now marked as legacy, and this specification no longer distinguishes between instance and device layers. Layers are enabled during instance creation, and are able to intercept all commands operating on that instance or any of its child objects. At the time this was marked as legacy, there were no known device-only layers and no compelling reason to create one.

Render Pass Functions: Superseded via version 2

Vulkan 1.2 introduced new versions of several render pass functions. These provide the same functionality as the Vulkan 1.0 functionality but with greater extensibility.

Appendix G: API Boilerplate

This appendix defines Vulkan API features that are infrastructure required for a complete functional description of Vulkan, but do not logically belong elsewhere in the Specification.

Vulkan Header Files

Vulkan is defined as an API in the C99 language. Khronos provides a corresponding set of header files for applications using the API, which may be used in either C or C++ code. The interface descriptions in the specification are the same as the interfaces defined in these header files, and both are derived from the `vk.xml` XML API Registry, which is the canonical machine-readable description of the Vulkan API. The Registry, scripts used for processing it into various forms, and documentation of the registry schema are available as described at <https://registry.khronos.org/vulkansc/#apiregistry>.

Language bindings for other languages can be defined using the information in the Specification and the Registry. Khronos does not provide any such bindings, but third-party developers have created some additional bindings.

Vulkan Combined API Header `vulkan_sc.h` (Informative)

Applications normally will include the header `vulkan_sc.h`. In turn, `vulkan_sc.h` always includes the following headers:

- `vk_platform.h`, defining platform-specific macros and headers.
- `vulkan_sc_core.h`, defining APIs for the Vulkan core and all registered extensions *other* than `window system-specific` and `provisional` extensions, which are included in separate header files.

In addition, specific preprocessor macros defined at the time `vulkan_sc.h` is included cause header files for the corresponding window system-specific and provisional interfaces to be included, as described below.

Vulkan Platform-Specific Header `vk_platform.h` (Informative)

Platform-specific macros and interfaces are defined in `vk_platform.h`. These macros are used to control platform-dependent behavior, and their exact definitions are under the control of specific platforms and Vulkan implementations.

Platform-Specific Calling Conventions

On many platforms the following macros are empty strings, causing platform- and compiler-specific default calling conventions to be used.

`VKAPI_ATTR` is a macro placed before the return type in Vulkan API function declarations. This macro controls calling conventions for C++11 and GCC/Clang-style compilers.

`VKAPI_CALL` is a macro placed after the return type in Vulkan API function declarations. This macro controls calling conventions for MSVC-style compilers.

VKAPI_PTR is a macro placed between the '(' and '*' in Vulkan API function pointer declarations. This macro also controls calling conventions, and typically has the same definition as **VKAPI_ATTR** or **VKAPI_CALL**, depending on the compiler.

With these macros, a Vulkan function declaration takes the form of:

```
VKAPI_ATTR <return_type> VKAPI_CALL <command_name>(<command_parameters>);
```

Additionally, a Vulkan function pointer type declaration takes the form of:

```
typedef <return_type> (VKAPI_PTR *PFN_<command_name>)(<command_parameters>);
```

Platform-Specific Header Control

If the **VK_NO_STDINT_H** macro is defined by the application at compile time, extended integer types used by the Vulkan API, such as **uint8_t**, **must** also be defined by the application. Otherwise, the Vulkan headers will not compile. If **VK_NO_STDINT_H** is not defined, the system **<stdint.h>** is used to define these types. There is a fallback path when Microsoft Visual Studio version 2008 and earlier versions are detected at compile time.

If the **VK_NO_STDDEF_H** macro is defined by the application at compile time, **size_t**, **must** also be defined by the application. Otherwise, the Vulkan headers will not compile. If **VK_NO_STDDEF_H** is not defined, the system **<stddef.h>** is used to define this type.

Vulkan Core API Header **vulkan_sc_core.h**

Applications that do not make use of window system-specific extensions may simply include **vulkan_sc_core.h** instead of **vulkan_sc.h**, although there is usually no reason to do so. In addition to the Vulkan API, **vulkan_sc_core.h** also defines and / or uses a small number of C preprocessor macros that are described below.

vulkan_sc_core.hpp provides the same functionality as **vulkan_sc_core.h**, but does so in a manner that is aligned for compliance with MISRA C++. In contrast, **vulkan_sc_core.h** is aligned for compliance with MISRA C:2012.

Vulkan Header File Compile Time Controls

If the **VK_NO_PROTOTYPES** macro is defined by an application at compile time, prototypes for Vulkan APIs will not be included. Only typedefs for API function pointers will be defined.

This is intended for applications using their own function loader and dispatch mechanism.

If the macro is not defined by the application, prototypes for Vulkan APIs will be included.

If the **VK_ONLY_EXPORTED_PROTOTYPES** macro is defined by an application at compile time, only prototypes for Vulkan APIs tagged as `"exported"` in the API XML will be included. For non-tagged APIs, only typedefs for API function pointers will be defined.

This is intended to match APIs which are statically exported by the Vulkan loader. At present, the exported APIs are only those defined by Vulkan core versions.

If the macro is not defined by the application, prototypes for all Vulkan APIs will be included.

Vulkan Header File Version Number

`VK_HEADER_VERSION` is the version number of the `vulkan_sc_core.h` header. This value is kept synchronized with the patch version of the released Specification.

```
// Provided by VK_VERSION_1_0
// Version of this file
#define VK_HEADER_VERSION 20
```

`VK_HEADER_VERSION_COMPLETE` is the complete version number of the `vulkan_sc_core.h` header, comprising the major, minor, and patch versions. The major/minor values are kept synchronized with the complete version of the released Specification. This value is intended for use by automated tools to identify exactly which version of the header was used during their generation.

Applications should not use this value as their `VkApplicationInfo::apiVersion`. Instead applications should explicitly select a specific fixed major/minor API version using, for example, one of the `VK_API_VERSION_*` values.

```
// Provided by VK_VERSION_1_0
// Complete version of this file
#define VK_HEADER_VERSION_COMPLETE VK_MAKE_API_VERSION(VKSC_API_VARIANT, 1, 0,
VK_HEADER_VERSION)
```

Vulkan Handle Macros

`VK_DEFINE_HANDLE` defines a [dispatchable handle](#) type.

```
// Provided by VK_VERSION_1_0
#define VK_DEFINE_HANDLE(object) typedef struct object##_T* (object);
```

- `object` is the name of the resulting C type.

The only dispatchable handle types are those related to device and instance management, such as `VkDevice`.

`VK_DEFINE_NON_DISPATCHABLE_HANDLE` defines a [non-dispatchable handle](#) type.


```
// Provided by VK_VERSION_1_0
#ifndef VK_DEFINE_NON_DISPATCHABLE_HANDLE
    #if (VK_USE_64_BIT_PTR_DEFINES==1)
        #define VK_DEFINE_NON_DISPATCHABLE_HANDLE(object) typedef struct object##_T
    *(object);
    #else
        #define VK_DEFINE_NON_DISPATCHABLE_HANDLE(object) typedef uint64_t (object);
    #endif
#endif
```

- **object** is the name of the resulting C type.

Most Vulkan handle types, such as [VkBuffer](#), are non-dispatchable.

NOTE

The `vulkan_sc_core.h` header allows the `VK_DEFINE_NON_DISPATCHABLE_HANDLE` and `VK_NULL_HANDLE` definitions to be overridden by the application. If `VK_DEFINE_NON_DISPATCHABLE_HANDLE` is already defined when `vulkan_sc_core.h` is compiled, the default definitions for `VK_DEFINE_NON_DISPATCHABLE_HANDLE` and `VK_NULL_HANDLE` are skipped. This allows the application to define a binary-compatible custom handle which **may** provide more type-safety or other features needed by the application. Applications **must** not define handles in a way that is not binary compatible - where binary compatibility is platform dependent.

`VK_NULL_HANDLE` is a reserved value representing a non-valid object handle. It may be passed to and returned from Vulkan commands only when [specifically allowed](#).

```
// Provided by VK_VERSION_1_0
#ifndef VK_DEFINE_NON_DISPATCHABLE_HANDLE
    #if (VK_USE_64_BIT_PTR_DEFINES==1)
        #if (defined(__cplusplus) && (__cplusplus >= 201103L)) || (defined(_MSVC_LANG)
&& (_MSVC_LANG >= 201103L))
            #define VK_NULL_HANDLE nullptr
        #else
            #define VK_NULL_HANDLE ((void*)0)
        #endif
    #else
        #define VK_NULL_HANDLE 0ULL
    #endif
#endif
#ifndef VK_NULL_HANDLE
    #define VK_NULL_HANDLE 0
#endif
```

`VK_USE_64_BIT_PTR_DEFINES` defines whether the default non-dispatchable handles are declared using either a 64-bit pointer type or a 64-bit unsigned integer type.

`VK_USE_64_BIT_PTR_DEFINES` is set to '1' to use a 64-bit pointer type or any other value to use a 64-bit

unsigned integer type.

```
// Provided by VK_VERSION_1_0
#ifndef VK_USE_64_BIT_PTR_DEFINES
    #if defined(__LP64__) || defined(_WIN64) || (defined(__x86_64__) &&
!defined(__ILP32__) ) || defined(_M_X64) || defined(__ia64) || defined (_M_IA64) ||
defined(__aarch64__) || defined(__powerpc64__) || (defined(__riscv) && __riscv_xlen ==
64)
        #define VK_USE_64_BIT_PTR_DEFINES 1
    #else
        #define VK_USE_64_BIT_PTR_DEFINES 0
    #endif
#endif
```

NOTE

The `vulkan_sc_core.h` header allows the `VK_USE_64_BIT_PTR_DEFINES` definition to be overridden by the application. This allows the application to select either a 64-bit pointer type or a 64-bit unsigned integer type for non-dispatchable handles in the case where the predefined preprocessor check does not identify the desired configuration.

Window System-Specific Header Control (Informative)

To use a Vulkan extension supporting a platform-specific window system, header files for that window system **must** be included at compile time, or platform-specific types **must** be forward-declared. The Vulkan header files are unable to determine whether or not an external header is available at compile time, so platform-specific extensions are provided in separate headers from the core API and platform-independent extensions, allowing applications to decide which ones they need to be defined and how the external headers are included.

Extensions dependent on particular sets of platform headers, or that forward-declare platform-specific types, are declared in a header named for that platform. Before including these platform-specific Vulkan headers, applications **must** include both `vulkan_sc_core.h` and any external native headers the platform extensions depend on.

As a convenience for applications that do not need the flexibility of separate platform-specific Vulkan headers, `vulkan_sc.h` includes `vulkan_sc_core.h`, and then conditionally includes platform-specific Vulkan headers and the external headers they depend on. Applications control which platform-specific headers are included by #defining macros before including `vulkan_sc.h`.

The correspondence between platform-specific extensions, external headers they require, the platform-specific header which declares them, and the preprocessor macros which enable inclusion by `vulkan_sc.h` are shown in the [following table](#).

Table 89. Window System Extensions and Headers

Extension Name	Window System Name	Platform-specific Header	Required External Headers	Controlling <code>vulkan_sc.h</code> Macro
VK_KHR_android_surface	Android	<code>vulkan_android.h</code>	None	VK_USE_PLATFORM_ANDROID_KHR
VK_KHR_wayland_surface	Wayland	<code>vulkan_wayland.h</code>	<code><wayland-client.h></code>	VK_USE_PLATFORM_WAYLAND_KHR
VK_KHR_win32_surface, VK_KHR_external_memory_win32, VK_KHR_win32_keyed_mutex, VK_KHR_external_semaphore_win32, VK_KHR_external_fence_win32, VK_NV_external_memory_win32, VK_NV_win32_keyed_mutex	Microsoft Windows	<code>vulkan_win32.h</code>	<code><windows.h></code>	VK_USE_PLATFORM_WIN32_KHR
VK_KHR_xcb_surface	X11 Xcb	<code>vulkan_xcb.h</code>	<code><xcb/xcb.h></code>	VK_USE_PLATFORM_XCB_KHR
VK_KHR_xlib_surface	X11 Xlib	<code>vulkan_xlib.h</code>	<code><X11/Xlib.h></code>	VK_USE_PLATFORM_XLIB_KHR
VK_EXT_directfb_surface	DirectFB	<code>vulkan_directfb.h</code>	<code><directfb/directfb.h></code>	VK_USE_PLATFORM_DIRECTFB_EXT
VK_EXT_acquire_xlib_display	X11 XRandR	<code>vulkan_xlib_xrandr.h</code>	<code><X11/Xlib.h></code> , <code><X11/extensions/Xrandr.h></code>	VK_USE_PLATFORM_XLIB_XRANDR_EXT
VK_GGP_stream_descriptor_surface, VK_GGP_frame_token	Google Games Platform	<code>vulkan_ggp.h</code>	<code><ggp_c/vulkan_types.h></code>	VK_USE_PLATFORM_GGP
VK_MVK_ios_surface	iOS	<code>vulkan_ios.h</code>	None	VK_USE_PLATFORM_IOS_MVK
VK_MVK_macos_surface	macOS	<code>vulkan_macos.h</code>	None	VK_USE_PLATFORM_MACOS_MVK
VK_OHOS_surface	OHOS	<code>vulkan_ohos.h</code>	None	VK_USE_PLATFORM_OHOS
VK_NN_vi_surface	VI	<code>vulkan_vi.h</code>	None	VK_USE_PLATFORM_VI_NN
VK_FUCHSIA_imagepipe_surface	Fuchsia	<code>vulkan_fuchsia.h</code>	<code><zircon/types.h></code>	VK_USE_PLATFORM_FUCHSIA
VK_EXT_metal_surface	Metal on CoreAnimation	<code>vulkan_metal.h</code>	None	VK_USE_PLATFORM_METAL_EXT
VK_QNX_screen_surface	QNX Screen	<code>vulkan_screen.h</code>	<code><screen/screen.h></code>	VK_USE_PLATFORM_SCREEN_QNX

Extension Name	Window System Name	Platform-specific Header	Required External Headers	Controlling <code>vulkan_sc.h</code> Macro
<code>VK_NV_external_sc_i_sync</code> , <code>VK_NV_external_sc_i_sync2</code> , <code>VK_NV_external_memory_sc_i_buf</code>	NVIDIA Sci	<code>vulkan_sc_i.h</code>	<code><nvscisync.h></code> , <code><nvscibuf.h></code>	<code>VK_USE_PLATFORM_SC_I</code>

NOTE

This section describes the purpose of the headers independently of the specific underlying functionality of the window system extensions themselves. Each extension name will only link to a description of that extension when viewing a specification built with that extension included.

Provisional Extension Header Control (Informative)

Provisional extensions **should** not be used in production applications. The functionality defined by such extensions **may** change in ways that break backwards compatibility between revisions, and before final release of a non-provisional version of that extension.

Provisional extensions are defined in a separate *provisional header*, `vulkan_beta.h`, allowing applications to decide whether or not to include them. The mechanism is similar to [window system-specific headers](#): before including `vulkan_beta.h`, applications **must** include `vulkan_sc_core.h`.

NOTE

Sometimes a provisional extension will include a subset of its interfaces in `vulkan_sc_core.h`. This may occur if the provisional extension is promoted from an existing vendor or EXT extension and some of the existing interfaces are defined as aliases of the provisional extension interfaces. All other interfaces of that provisional extension which are not aliased will be included in `vulkan_beta.h`.

As a convenience for applications, `vulkan_sc.h` conditionally includes `vulkan_beta.h`. Applications **can** control inclusion of `vulkan_beta.h` by #defining the macro `VK_ENABLE_BETA_EXTENSIONS` before including `vulkan_sc.h`.

NOTE

Starting in version 1.2.171 of the Specification, all provisional enumerants are protected by the macro `VK_ENABLE_BETA_EXTENSIONS`. Applications needing to use provisional extensions must always define this macro, even if they are explicitly including `vulkan_beta.h`. This is a minor change to behavior, affecting only provisional extensions.

NOTE

This section describes the purpose of the provisional header independently of the specific provisional extensions which are contained in that header at any given time. The extension appendices for provisional extensions note their provisional status, and link back to this section for more information. Provisional extensions are intended to provide early access for bleeding-edge developers, with the understanding that extension interfaces may change in response to developer feedback. Provisional extensions are very likely to eventually be updated and

released as non-provisional extensions, but there is no guarantee this will happen, or how long it will take if it does happen.

Appendix H: Invariance

The Vulkan specification is not pixel exact. It therefore does not guarantee an exact match between images produced by different Vulkan implementations. However, the specification does specify exact matches, in some cases, for images produced by the same implementation. The purpose of this appendix is to identify and provide justification for those cases that require exact matches.

Repeatability

The obvious and most fundamental case is repeated issuance of a series of Vulkan commands. For any given Vulkan and framebuffer state vector, and for any Vulkan command, the resulting Vulkan and framebuffer state **must** be identical whenever the command is executed on that initial Vulkan and framebuffer state. This repeatability requirement does not apply when using shaders containing side effects (image and buffer variable stores and atomic operations), because these memory operations are not guaranteed to be processed in a defined order.

One purpose of repeatability is avoidance of visual artifacts when a double-buffered scene is redrawn. If rendering is not repeatable, swapping between two buffers rendered with the same command sequence **may** result in visible changes in the image. Such false motion is distracting to the viewer. Another reason for repeatability is testability.

Repeatability, while important, is a weak requirement. Given only repeatability as a requirement, two scenes rendered with one (small) polygon changed in position might differ at every pixel. Such a difference, while within the law of repeatability, is certainly not within its spirit. Additional invariance rules are desirable to ensure useful operation.

Multi-Pass Algorithms

Invariance is necessary for a whole set of useful multi-pass algorithms. Such algorithms render multiple times, each time with a different Vulkan mode vector, to eventually produce a result in the framebuffer. Examples of these algorithms include:

- “Erasing” a primitive from the framebuffer by redrawing it, either in a different color or using the XOR logical operation.
- Using stencil operations to compute capping planes.

Invariance Rules

For a given Vulkan device:

Rule 1 *For any given Vulkan and framebuffer state vector, and for any given Vulkan command, the resulting Vulkan and framebuffer state **must** be identical each time the command is executed on that initial Vulkan and framebuffer state.*

Rule 2 *Changes to the following state values have no side effects (the use of any other state value is not affected by the change):*

Required:

- *Color and depth/stencil attachment contents*
- *Scissor parameters (other than enable)*
- *Write masks (color, depth, stencil)*
- *Clear values (color, depth, stencil)*

Strongly suggested:

- *Stencil parameters (other than enable)*
- *Depth test parameters (other than enable)*
- *Blend parameters (other than enable)*
- *Logical operation parameters (other than enable)*

Corollary 1 *Fragment generation is invariant with respect to the state values listed in Rule 2.*

Rule 3 *The arithmetic of each per-fragment operation is invariant except with respect to parameters that directly control it.*

Corollary 2 *Images rendered into different color attachments of the same framebuffer, either simultaneously or separately using the same command sequence, are pixel identical.*

Rule 4 *Identical pipelines will produce the same result when run multiple times with the same input. The wording “Identical pipelines” means [VkPipeline](#) objects that have been created with identical SPIR-V binaries and identical state, which are then used by commands executed using the same Vulkan state vector. Invariance is relaxed for shaders with side effects, such as performing stores or atomics.*

Rule 5 *All fragment shaders that either conditionally or unconditionally assign [FragCoord.z](#) to [FragDepth](#) are depth-invariant with respect to each other, for those fragments where the assignment to [FragDepth](#) actually is done.*

If a sequence of Vulkan commands specifies primitives to be rendered with shaders containing side effects (image and buffer variable stores and atomic operations), invariance rules are relaxed. In particular, rule 1, corollary 2, and rule 4 do not apply in the presence of shader side effects.

The following weaker versions of rules 1 and 4 apply to Vulkan commands involving shader side effects:

Rule 6 *For any given Vulkan and framebuffer state vector, and for any given Vulkan command, the contents of any framebuffer state not directly or indirectly affected by results of shader image or buffer variable stores or atomic operations **must** be identical each time the command is executed on that initial Vulkan and framebuffer state.*

Rule 7 *Identical pipelines will produce the same result when run multiple times with the same input as long as:*

- *shader invocations do not use image atomic operations;*

- *no framebuffer memory is written to more than once by image stores, unless all such stores write the same value; and*
- *no shader invocation, or other operation performed to process the sequence of commands, reads memory written to by an image store.*

NOTE

The OpenGL specification has the following invariance rule: Consider a primitive p' obtained by translating a primitive p through an offset (x, y) in window coordinates, where x and y are integers. As long as neither p' nor p is clipped, it **must** be the case that each fragment f' produced from p' is identical to a corresponding fragment f from p except that the center of f' is offset by (x, y) from the center of f .

This rule does not apply to Vulkan and is an intentional difference from OpenGL.

When any sequence of Vulkan commands triggers shader invocations that perform image stores or atomic operations, and subsequent Vulkan commands read the memory written by those shader invocations, these operations **must** be explicitly synchronized.

Tessellation Invariance

When using a pipeline containing tessellation evaluation shaders, the fixed-function tessellation primitive generator consumes the input patch specified by an application and emits a new set of primitives. The following invariance rules are intended to provide repeatability guarantees. Additionally, they are intended to allow an application with a carefully crafted tessellation evaluation shader to ensure that the sets of triangles generated for two adjacent patches have identical vertices along shared patch edges, avoiding “cracks” caused by minor differences in the positions of vertices along shared edges.

Rule 1 *When processing two patches with identical outer and inner tessellation levels, the tessellation primitive generator will emit an identical set of point, line, or triangle primitives as long as the pipeline used to process the patch primitives has tessellation evaluation shaders specifying the same tessellation mode, spacing, vertex order, and point mode decorations. Two sets of primitives are considered identical if and only if they contain the same number and type of primitives and the generated tessellation coordinates for the vertex numbered m of the primitive numbered n are identical for all values of m and n .*

Rule 2 *The set of vertices generated along the outer edge of the subdivided primitive in triangle and quad tessellation, and the tessellation coordinates of each, depend only on the corresponding outer tessellation level and the spacing decorations in the tessellation shaders of the pipeline.*

Rule 3 *The set of vertices generated when subdividing any outer primitive edge is always symmetric. For triangle tessellation, if the subdivision generates a vertex with tessellation coordinates of the form $(0, x, 1-x)$, $(x, 0, 1-x)$, or $(x, 1-x, 0)$, it will also generate a vertex with coordinates of exactly $(0, 1-x, x)$, $(1-x, 0, x)$, or $(1-x, x, 0)$, respectively. For quad tessellation, if the subdivision generates a vertex with coordinates of $(x, 0)$ or $(0, x)$, it will also generate a vertex with coordinates of exactly $(1-x, 0)$ or $(0, 1-x)$, respectively. For isoline tessellation, if it generates vertices at $(0, x)$ and $(1, x)$ where x is not zero, it will also generate vertices at exactly $(0, 1-x)$ and $(1, 1-x)$, respectively.*

Rule 4 *The set of vertices generated when subdividing outer edges in triangular and quad tessellation*

must be independent of the specific edge subdivided, given identical outer tessellation levels and spacing. For example, if vertices at $(x, 1 - x, 0)$ and $(1 - x, x, 0)$ are generated when subdividing the $w = 0$ edge in triangular tessellation, vertices **must** be generated at $(x, 0, 1 - x)$ and $(1 - x, 0, x)$ when subdividing an otherwise identical $v = 0$ edge. For quad tessellation, if vertices at $(x, 0)$ and $(1 - x, 0)$ are generated when subdividing the $v = 0$ edge, vertices **must** be generated at $(0, x)$ and $(0, 1 - x)$ when subdividing an otherwise identical $u = 0$ edge.

Rule 5 When processing two patches that are identical in all respects enumerated in rule 1 except for vertex order, the set of triangles generated for triangle and quad tessellation **must** be identical except for vertex and triangle order. For each triangle $n1$ produced by processing the first patch, there **must** be a triangle $n2$ produced when processing the second patch each of whose vertices has the same tessellation coordinates as one of the vertices in $n1$.

Rule 6 When processing two patches that are identical in all respects enumerated in rule 1 other than matching outer tessellation levels and/or vertex order, the set of interior triangles generated for triangle and quad tessellation **must** be identical in all respects except for vertex and triangle order. For each interior triangle $n1$ produced by processing the first patch, there **must** be a triangle $n2$ produced when processing the second patch each of whose vertices has the same tessellation coordinates as one of the vertices in $n1$. A triangle produced by the tessellator is considered an interior triangle if none of its vertices lie on an outer edge of the subdivided primitive.

Rule 7 For quad and triangle tessellation, the set of triangles connecting an inner and outer edge depends only on the inner and outer tessellation levels corresponding to that edge and the spacing decorations.

Rule 8 The value of all defined components of **TessCoord** will be in the range $[0, 1]$. Additionally, for any defined component x of **TessCoord**, the results of computing $1.0 - x$ in a tessellation evaluation shader will be exact. If any floating-point values in the range $[0, 1]$ fail to satisfy this property, such values **must** not be used as tessellation coordinate components.

Appendix I: Vulkan SC Deviations From Base Vulkan

Additions

The following extensions have been added to Vulkan SC:

Extension	Level
VK_KHR_object_refresh	Optional

The following items have been added to Vulkan SC:

Chapter	Additions
Fundamentals	- extending VkResult VK_ERROR_VALIDATION_FAILED [SCID-1] VK_ERROR_INVALID_PIPELINE_CACHE_DATA [SCID-1] VK_ERROR_NO_PIPELINE_MATCH [SCID-1]
Devices and Queues	VkPhysicalDeviceVulkanSC10Properties [SCID-1] VkDeviceObjectReservationCreateInfo [SCID-4] VkPerformanceQueryReservationInfoKHR [SCID-4] VkPipelinePoolSize [SCID-4]
Command Buffers	VkCommandPoolMemoryReservationCreateInfo [SCID-4] vkGetCommandPoolMemoryConsumption [SCID-1] VkCommandPoolMemoryConsumption [SCID-1]
Pipelines	- extending VkPipelineCacheCreateFlagBits VK_PIPELINE_CACHE_CREATE_READ_ONLY_BIT [SCID-1], [SCID-8] VK_PIPELINE_CACHE_CREATE_USE_APPLICATION_STORAGE_BIT [SCID-2] - extending VkPipelineCacheHeaderVersion VK_PIPELINE_CACHE_HEADER_VERSION_SAFETY_CRITICAL_ONE [SCID-1], [SCID-8] VkPipelineCacheHeaderVersionSafetyCriticalOne [SCID-1], [SCID-8] VkPipelineCacheValidationVersion [SCID-1], [SCID-8] VkPipelineCacheSafetyCriticalIndexEntry [SCID-1], [SCID-8] VkPipelineCacheStageValidationIndexEntry [SCID-1], [SCID-8] VkPipelineOfflineCreateInfo [SCID-1], [SCID-8] VkPipelineMatchControl [SCID-1]

Chapter	Additions
Memory Allocation	- extending VkMemoryHeapFlagBits VK_MEMORY_HEAP_SEU_SAFE_BIT [SCID-1]
Features	VkPhysicalDeviceVulkanSC10Features [SCID-1]
Debugging	VkFaultData [SCID-6] VkFaultCallbackInfo [SCID-6] VkFaultLevel [SCID-6] VkFaultType [SCID-6] VkFaultQueryBehavior [SCID-6] PFN_vkFaultCallbackFunction [SCID-6] vkGetFaultData [SCID-6]

Modifications

The following aspects of Base Vulkan have been modified for Vulkan SC:

Chapter	Modifications
Fundamentals	If VkPhysicalDeviceVulkanSC10Properties::deviceNoDynamicHostAllocations is VK_TRUE, VK_ERROR_OUT_OF_HOST_MEMORY must not be returned by physical or logical device commands which explicitly disallow it [SCID-4].
Devices and Queues	- The VkDeviceCreateInfo::pNext chain must include a VkDeviceObjectReservationCreateInfo structure [SCID-4]. - The VkDeviceCreateInfo::pNext chain must include a VkPhysicalDeviceVulkanSC10Features structure [SCID-1]. vkCreateDevice returns VK_ERROR_INVALID_PIPELINE_CACHE_DATA if the pInitialData member of any element of VkDeviceObjectReservationCreateInfo::pPipelineCacheCreateInfos is a pointer to incompatible pipeline cache data [SCID-1]. vkCreateDevice returns VK_ERROR_INVALID_PIPELINE_CACHE_DATA if the pInitialData member of any element of VkDeviceObjectReservationCreateInfo::pPipelineCacheCreateInfos is not aligned according to the platform and/or implementation specific pipeline cache data alignment requirements [SCID-8].

Chapter	Modifications
Command Buffers	• The <code>VkCommandPoolCreateInfo::pNext</code> chain must include a valid <code>VkCommandPoolMemoryReservationCreateInfo</code> structure [SCID-4]. • If <code>commandPoolResetCommandBuffer</code> is not supported [SCID-8], <code>vkResetCommandBuffer</code> must not be called. • <code>vkFreeCommandBuffers</code> does not return the memory used by command recording back to its parent command pool [SCID-4]. This memory is reclaimed when <code>vkResetCommandPool</code> is next called. • If <code>VkPhysicalDeviceVulkanSC10Properties::commandPoolMultipleCommandBuffersRecording</code> is <code>VK_FALSE</code>, then only one command buffer from a command pool can be in the <code>recording state</code> at a time [SCID-8]. • If <code>VkPhysicalDeviceVulkanSC10Properties::commandBufferSimultaneousUse</code> is <code>VK_FALSE</code>, then <code>VkCommandBufferBeginInfo::flags</code> must not include <code>VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT</code> [SCID-8]. • If <code>commandPoolResetCommandBuffer</code> is not supported, <code>commandBuffer</code> must be in the <code>initial state</code> when <code>vkBeginCommandBuffer</code> is called [SCID-8]. • If <code>VkPhysicalDeviceVulkanSC10Properties::secondaryCommandBufferNullOrImagelessFramebuffer</code> is <code>VK_FALSE</code>, then <code>VkCommandBufferInheritanceInfo::framebuffer</code> must not be <code>VK_NULL_HANDLE</code> and must not have been created with a <code>VkFramebufferCreateInfo::flags</code> value that includes <code>VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT</code> if the command buffer will be executed within a render pass instance [SCID-8].

Chapter	Modifications
Pipelines	• <code>vkCreateComputePipelines</code> returns <code>VK_ERROR_NO_PIPELINE_MATCH</code> if the <code>VkComputePipelineCreateInfo::pNext</code> chain does not include a valid <code>VkPipelineOfflineCreateInfo</code> structure [SCID-1]. • <code>vkCreateComputePipelines::pipelineCache</code> must not be <code>VK_NULL_HANDLE</code> [SCID-1], [SCID-8]. • <code>VkComputePipelineCreateInfo::basePipelineHandle</code> must be <code>VK_NULL_HANDLE</code> [SCID-8]. • <code>VkComputePipelineCreateInfo::basePipelineIndex</code> must be zero [SCID-8]. • <code>vkCreateGraphicsPipelines</code> returns <code>VK_ERROR_NO_PIPELINE_MATCH</code> if the <code>VkGraphicsPipelineCreateInfo::pNext</code> chain does not include a valid <code>VkPipelineOfflineCreateInfo</code> structure [SCID-1]. • <code>vkCreateGraphicsPipelines::pipelineCache</code> must not be <code>VK_NULL_HANDLE</code> [SCID-1], [SCID-8]. • <code>VkGraphicsPipelineCreateInfo::basePipelineHandle</code> must be <code>VK_NULL_HANDLE</code> [SCID-8]. • <code>VkGraphicsPipelineCreateInfo::basePipelineIndex</code> must be zero [SCID-8]. • <code>VkPipelineCacheCreateInfo::pInitialData</code> must point to a valid pipeline cache that has been generated offline [SCID-1], [SCID-8]. • <code>VkPipelineCacheCreateInfo::initialDataSize</code> must not be 0 [SCID-1], [SCID-8]. • <code>VkPipelineCacheCreateInfo::pInitialData</code> must not be <code>NULL</code> [SCID-1], [SCID-8]. • <code>VkPipelineCacheCreateInfo::flags</code> must include <code>VK_PIPELINE_CACHE_CREATE_READ_ONLY_BIT</code> [SCID-1], [SCID-8]. • <code>VkPipelineCacheCreateInfo::flags</code> must include <code>VK_PIPELINE_CACHE_CREATE_USE_APPLICATION_STORAGE_BIT</code> [SCID-2]. • The contents of <code>VkPipelineCacheCreateInfo</code>, including the data pointed to by <code>VkPipelineCacheCreateInfo::pInitialData</code>, passed to <code>vkCreatePipelineCache</code> must be the same as specified in one of the <code>VkDeviceObjectReservationCreateInfo::pPipelineCacheCreateInfos</code> structures when the device was created [SCID-1]. • <code>VkPipelineCacheHeaderVersionOne::headerSize</code> must be 56 [SCID-1]. • <code>VkPipelineCacheHeaderVersionOne::headerVersion</code> must be <code>VK_PIPELINE_CACHE_HEADER_VERSION_SAFETY_CRITICAL_ONE</code> [SCID-1].

Chapter	Modifications
Memory Allocation	• <code>vkCreate*::pAllocator</code> must be <code>NULL</code> [SCID-2], [SCID-8]. • <code>vkDestroy*::pAllocator</code> must be <code>NULL</code> [SCID-2], [SCID-8]. • <code>vk*Memory::pAllocator</code> must be <code>NULL</code> [SCID-2], [SCID-8]. • <code>vkRegisterDeviceEventEXT::pAllocator</code> must be <code>NULL</code> [SCID-8].
Resource Creation	• <code>VkBufferCreateInfo::flags</code> must not contain any of the <code>VK_BUFFER_CREATE_SPARSE_BINDING_BIT</code>, <code>VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT</code>, or <code>VK_BUFFER_CREATE_SPARSE_ALIASED_BIT</code> flags [SCID-8]. • <code>VkImageCreateInfo::flags</code> must not contain any of the <code>VK_IMAGE_CREATE_SPARSE_BINDING_BIT</code>, <code>VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT</code>, <code>VK_IMAGE_CREATE_SPARSE_ALIASED_BIT</code>, or <code>VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT</code> flags [SCID-8]. • <code>VkBindImageMemoryDeviceGroupInfo::splitInstanceBindRegionCount</code> must be zero [SCID-8].
Resource Descriptors	• If <code>recycleDescriptorSetMemory</code> is <code>VK_FALSE</code>, then freeing a descriptor set does not make the pool memory it used available to be reallocated until the descriptor pool is reset [SCID-4].
Sparse Resources	• <code>VkPhysicalDeviceSparseProperties::residencyStandard2DBlockShape</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceSparseProperties::residencyStandard2DMultisampleBlockShape</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceSparseProperties::residencyStandard3DBlockShape</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceSparseProperties::residencyAlignedMipSize</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceSparseProperties::residencyNonResidentStrict</code> must be reported as <code>VK_FALSE</code> [SCID-8].
WSI Swapchain	• <code>VkSwapchainCreateInfoKHR::flags</code> must not contain <code>VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR</code> [SCID-8]. • <code>VkSwapchainCreateInfoKHR::oldSwapchain</code> must be <code>VK_NULL_HANDLE</code> [SCID-4].

Chapter	Modifications
Features	• <code>VkPhysicalDeviceFeatures::shaderResourceResidency</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceFeatures::sparseBinding</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceFeatures::sparseResidencyBuffer</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceFeatures::sparseResidencyImage2D</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceFeatures::sparseResidencyImage3D</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceFeatures::sparseResidency2Samples</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceFeatures::sparseResidency4Samples</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceFeatures::sparseResidency8Samples</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceFeatures::sparseResidency16Samples</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceFeatures::sparseResidencyAliased</code> must be reported as <code>VK_FALSE</code> [SCID-8]. • <code>VkPhysicalDeviceVulkanSC10Features::shaderAtomicInstructions</code> are made optional [SCID-1]. • <code>VkPhysicalDeviceVulkan11Features::multiview</code> is made optional [SCID-8]. • <code>VkPhysicalDeviceVulkan12Features::timelineSemaphore</code> is made optional [SCID-8]. • <code>VkPhysicalDeviceVulkan12Features::vulkanMemoryModel</code> must be reported as <code>VK_TRUE</code> [SCID-1].
Limits	• <code>VkPhysicalDeviceLimits::maxFramebufferLayers</code> may be 1 if neither <code>geometryShader</code> or <code>shaderOutputLayer</code> are supported [SCID-8]. • <code>VkPhysicalDeviceVulkan12Properties::supportedDepthResolveModes</code> may be only <code>VK_RESOLVE_MODE_NONE</code> [SCID-8]. • <code>VkPhysicalDeviceVulkan12Properties::supportedStencilResolveModes</code> may be only <code>VK_RESOLVE_MODE_NONE</code> [SCID-8].

Removals

The following functionality has been removed from Base Vulkan in Vulkan SC:

Chapter	Removals
Fundamentals	• VkStructureType (legacy aliases) ◦ VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VARIABLE_POINTER_FEATURES [SCID-8] ◦ VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SHADER_DRAW_PARAMETER_FEATURES [SCID-8] ◦ VK_STRUCTURE_TYPE_SURFACE_CAPABILITIES2_EXT [SCID-8]
Devices and Queues	• VkQueueFlagBits ◦ VK_QUEUE_SPARSE_BINDING_BIT [SCID-8]
Command Buffers	• vkDestroyCommandPool [SCID-4] • vkTrimCommandPool, vkTrimCommandPoolKHR [SCID-8] • VkCommandPoolTrimFlags, VkCommandPoolTrimFlagsKHR [SCID-8] • VkCommandPoolResetFlagBits ◦ VK_COMMAND_POOL_RESET_RELEASE_RESOURCES_BIT [SCID-4]
Synchronization and Cache Control	• vkDestroySemaphoreSciSyncPoolNV [SCID-4]
Shaders	• VkStructureType ◦ VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO [SCID-8] • VkObjectType ◦ VK_OBJECT_TYPE_SHADER_MODULE [SCID-8] • vkCreateShaderModule, vkDestroyShaderModule [SCID-8] • VkShaderModule, VkShaderModuleCreateInfo [SCID-8] • VkShaderModuleCreateFlags [SCID-8] • VkShaderModuleCreateFlagBits [SCID-8]
Pipelines	• VkPipelineCreateFlagBits ◦ VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT [SCID-8] ◦ VK_PIPELINE_CREATE_DERIVATIVE_BIT [SCID-8] • vkMergePipelineCaches, vkGetPipelineCacheData [SCID-1], [SCID-8]
Memory Allocation	• VkSystemAllocationScope, VkInternalAllocationType [SCID-8] • vkFreeMemory [SCID-4]

Chapter	Removals
Resource Descriptors	• vkDestroyDescriptorPool [SCID-4] • VkStructureType ◦ VK_STRUCTURE_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_CREATE_INFO_KHR [SCID-8] • VkObjectType ◦ VK_OBJECT_TYPE_DESCRIPTOR_UPDATE_TEMPLATE_KHR [SCID-8] • vkCreateDescriptorUpdateTemplateKHR, vkDestroyDescriptorUpdateTemplateKHR, vkUpdateDescriptorSetWithTemplateKHR, vkCmdPushDescriptorSetWithTemplateKHR [SCID-8] • VkDescriptorUpdateTemplateKHR, VkDescriptorUpdateTemplateEntryKHR, VkDescriptorUpdateTemplateCreateInfoKHR [SCID-8] • VkDescriptorUpdateTemplateTypeKHR [SCID-8] • VkDescriptorUpdateTemplateCreateFlagsKHR [SCID-8] • VkDescriptorUpdateTemplateType ◦ VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_DESCRIPTOR_SET_KHR [SCID-8] ◦ VK_DESCRIPTOR_UPDATE_TEMPLATE_TYPE_PUSH_DESCRIPTORS [SCID-8]
Queries	• vkDestroyQueryPool [SCID-4]
Fragment Operations	• VkStencilFaceFlagBits (legacy alias) ◦ VK_STENCIL_FRONT_AND_BACK [SCID-8]

Chapter	Removals
Sparse Resources	• VkStructureType ◦ VK_STRUCTURE_TYPE_BIND_SPARSE_INFO [SCID-8] ◦ VK_STRUCTURE_TYPE_DEVICE_GROUP_BIND_SPARSE_INFO [SCID-8] ◦ VK_STRUCTURE_TYPE_IMAGE_SPARSE_MEMORY_REQUIREMENTS_INFO_2 [SCID-8] ◦ VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_SPARSE_IMAGE_FORMAT_INFO_2 [SCID-8] ◦ VK_STRUCTURE_TYPE_SPARSE_IMAGE_FORMAT_PROPERTIES_2 [SCID-8] ◦ VK_STRUCTURE_TYPE_SPARSE_IMAGE_MEMORY_REQUIREMENTS_2 [SCID-8] • VkSparseImageFormatProperties [SCID-8] • VkSparseImageFormatFlagBits [SCID-8] • VkSparseImageFormatFlags [SCID-8] • vkGetPhysicalDeviceSparseImageFormatProperties [SCID-8] • vkGetPhysicalDeviceSparseImageFormatProperties2 [SCID-8] • VkPhysicalDeviceSparseImageFormatInfo2 [SCID-8] • VkSparseImageFormatProperties2 [SCID-8] • VkSparseImageMemoryRequirements [SCID-8] • vkGetImageSparseMemoryRequirements [SCID-8] • vkGetImageSparseMemoryRequirements2 [SCID-8] • VkImageSparseMemoryRequirementsInfo2 [SCID-8] • VkSparseImageMemoryRequirements2 [SCID-8] • VkSparseMemoryBind [SCID-8] • VkSparseMemoryBindFlagBits [SCID-8] • VkSparseMemoryBindFlags [SCID-8] • VkSparseBufferMemoryBindInfo [SCID-8] • VkSparseImageOpaqueMemoryBindInfo [SCID-8] • VkSparseImageMemoryBindInfo [SCID-8] • VkSparseImageMemoryBind [SCID-8] • vkQueueBindSparse [SCID-8] • VkBindSparseInfo [SCID-8] • VkDeviceGroupBindSparseInfo [SCID-8]

Chapter	Removals
Window System Integration	• VkColorSpaceKHR (legacy aliases) ◦ VK_COLORSPACE_SRGB_NONLINEAR_KHR [SCID-8] ◦ VK_COLOR_SPACE_DCI_P3_LINEAR_EXT [SCID-8] • VkColorSpaceKHR (not well-defined) ◦ VK_COLOR_SPACE_DOLBYVISION_EXT [SCID-8]
WSI Swapchain	• vkDestroySwapchainKHR [SCID-4]

Extension Support

Vulkan SC supports a subset of the extensions supported in Base Vulkan. This subset was decided by:

- Excluding any extensions that would pose significant difficulty to certify their implementations.
- Excluding any extension that would not be used in deployed devices. This was primarily extensions focused on application development and debug.
- Excluding any extensions that are specific to an Operating System or Windowing system that is highly unlikely to be used in the Safety Critical space.
- Non-KHR or EXT extension are supported on request.

NOTE

During development it is likely that application developers will need additional functionality in a Vulkan SC implementation beyond what is provided by the supported extensions. This can be achieved by implementing a development focused version of the implementation that exposes additional Vulkan extensions and tools support but is non-conformant to the Vulkan SC specification.

A Vulkan SC conformant implementation with this additional functionality removed will be used on the end device.

Fault and Error Handling

Vulkan SC maintains the use of [VkResult Return Codes](#) on a small number of commands. These allow the command to confirm it completed successfully or return an error code for situations where a failure could be detected at runtime during the execution of the command.

In addition to [VkResult Return Codes](#) Vulkan SC adds [Fault Handling](#) support. This provides the implementation the ability to communicate information on errors or faults to the application that have been detected but are not covered by [VkResult Return Codes](#) in the Vulkan SC API. These could be runtime failures of the system or application faults that are detected asynchronously to the Vulkan API commands.

Undefined Behavior in the API

If an application uses the API incorrectly the behavior of the API is undefined. The Vulkan SC runtime will perform minimal error and state checking and it is assumed that applications are using the API correctly, see [Valid Usage](#).

With incorrect input to the API, the implementation could continue to function correctly, generate unexpected output, become unstable, or be terminated. The exact behavior will vary and be dependent on the specifics of the invalid usage and the implementation.

It is primarily the application's responsibility to ensure it always uses the API correctly. Potential methods to detect incorrect API usage include performing manual code inspection, use of validation layers during development, use of validation layers at runtime, or adding runtime checking to the application. Outside of this, Vulkan SC implementations **can** add implementation-specific targeted checks to detect invalid API usage that could significantly impact the correct operation of the application or implementation. The [Fault Handling](#) extension allows implementations to communicate information on such occurrences.

MISRA C:2012 Deviations

`vulkan_sc_core.h` is intended to be compatible with safety coding standards like MISRA C:2012.

The following provides information on items a MISRA C code analysis tool **may** report for a project using Vulkan SC.

MISRA headline guidelines are copyright © The MISRA Consortium Limited and are reproduced with permission. For further explanation of the directives and rules please see the *MISRA C:2012* specification (<https://www.misra.org.uk/misra-c/>). See *MISRA Compliance:2020* (<https://www.misra.org.uk/app/uploads/2021/06/MISRA-Compliance-2020.pdf>) for a framework for handling deviations.

Directives

Directive	4.6: "typedefs that indicate size and signedness should be used in place of the basic numerical types"
Category	Advisory
Note	This is reported for every <code>char</code> and <code>float</code> variable used in the API.
Rationale	Vulkan SC maintains the Base Vulkan type conventions for compatibility between APIs.

Rules

Rule	2.3: "A project should not contain unused type declarations"
Category	Advisory
Note	This is reported for any unused type definitions.

Rule	2.3: "A project should not contain unused type declarations"
Rationale	The <code>vulkan_sc_core.h</code> provides a complete API definition and it is expected that an application may not use all the provided type declarations.

Rule	2.4: "A project should not contain unused tag declarations"
Category	Advisory
Note	This is reported for each instance of <code>typedef struct VkStruct { ... } VkStruct;</code> and <code>typedef enum VkEnum { ... } VkEnum;</code> where the tag declaration is unused.
Rationale	The <code>vulkan_sc_core.h</code> provides a complete API definition and it is expected that an application may not use all the provided tag declarations. Vulkan SC maintains the Base Vulkan type conventions for compatibility between APIs. Tag declarations are required in case an application wishes to make forward declarations to API-defined types.

Rule	2.5: "A project should not contain unused macro declarations"
Category	Advisory
Note	This is reported for every unused macro defined in the header.
Rationale	The <code>vulkan_sc_core.h</code> provides a complete API definition and it is expected that an application may not use all the provided macro declarations.

Rule	5.1: "External identifiers shall be distinct"
Category	Required
Note	This is reported for identifiers with names that do not differ in the first 31 characters, such as <code>vkGetPhysicalDeviceFormatProperties</code> and <code>vkGetPhysicalDeviceFormatProperties2</code> .
Rationale	Vulkan SC maintains the Base Vulkan naming conventions for compatibility between APIs. Vulkan SC applications must be built using a compiler that treats enough characters as significant.

Rule	5.2: "Identifiers declared in the same <i>scope</i> and name space shall be distinct"
Category	Required
Note	This is reported for many <code>typedef</code> statements with long identifiers.
Rationale	Vulkan SC maintains the Base Vulkan type and naming conventions for compatibility between APIs. Vulkan SC applications must be built using a compiler that treats enough characters as significant.

Rule	5.4: "Macro identifiers shall be distinct"
Category	Required
Note	This is reported for macros with names that do not differ in the first 31 characters, such as <code>VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT</code> and <code>VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT</code> .

Rule	5.4: "Macro identifiers shall be distinct"
Rationale	Vulkan SC maintains the Base Vulkan naming conventions for compatibility between APIs. Vulkan SC applications must be built using a compiler that treats enough characters as significant.

Rule	8.6: "An identifier with external linkage shall have exactly one external definition"
Category	Required
Note	This is reported for every API command declaration, and the external definitions are provided by the implementation.
Rationale	It is expected that a Vulkan SC application will link against an implementation that provides these definitions.

Rule	19.2: "The <i>union</i> keyword should not be used"
Category	Advisory
Note	This is reported on the VkClearColorValue , VkClearValue , and VkPerformanceCounterResultKHR unions.
Rationale	These are required to remain compatible with the Base Vulkan API.

Rule	20.1: "#include directives should only be preceded by preprocessor directives or comments"
Category	Advisory
Note	This is reported because the entire Vulkan SC API definition is wrapped in an extern "C" block.
Rationale	This is expected because the Vulkan SC API is a C ABI and the header may be included from C++ code.

Rule	20.10: "The # and ## preprocessor operators should not be used"
Category	Advisory
Note	This is reported for the two lines: <pre>#define VK_DEFINE_HANDLE(object) typedef struct object##_T* (object); #define VK_DEFINE_NON_DISPATCHABLE_HANDLE(object) typedef struct object##_T *(object);</pre>
Rationale	This is expected usage of the macro expansion operation and there are not multiple operators used in the statement.

Appendix J: Lexicon

This appendix defines terms, abbreviations, and API prefixes used in the Specification.

Glossary

The terms defined in this section are used consistently throughout the Specification and may be used with or without capitalization.

Accessible (Descriptor Binding)

A descriptor binding is accessible to a shader stage if that stage is included in the `stageFlags` of the descriptor binding. Descriptors using that binding **can** only be used by stages in which they are accessible.

Acquire Operation (Resource)

An operation that acquires ownership of an image subresource or buffer range.

Adjacent Vertex

A vertex in an adjacency primitive topology that is not part of a given primitive, but is accessible in geometry shaders.

Advanced Blend Operation

Blending performed using one of the blend operation enums introduced by the `VK_EXT_blend_operation_advanced` extension. See [Advanced Blending Operations](#).

Alias (API type/command)

An identical definition of another API type/command with the same behavior but a different name.

Aliased Range (Memory)

A range of a device memory allocation that is bound to multiple resources simultaneously.

Allocation Scope

An association of a host memory allocation to a parent object or command, where the allocation's lifetime ends before or at the same time as the parent object is freed or destroyed, or during the parent command.

API command

Any command defined in the Vulkan specification. These entry points all have a `vk` prefix.

Aspect (Image)

Some image types contain multiple kinds (called “aspects”) of data for each pixel, where each aspect is used in a particular way by the pipeline and **may** be stored differently or separately from other aspects. For example, the color components of an image format make up the color aspect of the image, and **can** be used as a framebuffer color attachment. Some operations, like depth testing, operate only on specific aspects of an image.

Attachment (Render Pass)

A zero-based integer index name used in render pass creation to refer to a framebuffer attachment that is accessed by one or more subpasses. The index also refers to an attachment description which includes information about the properties of the image view that will later be attached.

Availability Operation

An operation that causes the values generated by specified memory write accesses to become available for future access.

Available

A state of values written to memory that allows them to be made visible.

Back-Facing

See Facingness.

Batch

A single structure submitted to a queue as part of a [queue submission command](#), describing a set of queue operations to execute.

Backwards Compatibility

A given version of the API is backwards compatible with an earlier version if an application, relying only on valid behavior and functionality defined by the earlier specification, is able to correctly run against each version without any modification. This assumes no active attempt by that application to not run when it detects a different version.

Binary Semaphore

A semaphore with a boolean payload indicating whether the semaphore is signaled or unsignaled. Represented by a [VkSemaphore](#) object created with a semaphore type of `VK_SEMAPHORE_TYPE_BINARY`.

Binding (Memory)

An association established between a range of a resource object and a range of a memory object. These associations determine the memory locations affected by operations performed on elements of a resource object. Memory bindings are established using the [vkBindBufferMemory](#) command for non-sparse buffer objects, and using the [vkBindImageMemory](#) command for non-sparse image objects.

Blend Constant

Four floating-point (RGBA) values used as an input to blending.

Blending

Arithmetic operations between a fragment color value and a value in a color attachment that produce a final color value to be written to the attachment.

Buffer

A resource that represents a linear array of data in device memory. Represented by a [VkBuffer](#) object.

Buffer Device Address

A 64-bit value used in a shader to access buffer memory through the `PhysicalStorageBuffer` storage class.

Buffer View

An object that represents a range of a specific buffer, and state controlling how the contents are interpreted. Represented by a `VkBufferView` object.

Built-In Variable

A variable decorated in a shader, where the decoration makes the variable take values provided by the execution environment or values that are generated by fixed-function pipeline stages.

Built-In Interface Block

A block defined in a shader containing only variables decorated with built-in decorations, and is used to match against other shader stages.

Clip Coordinates

The homogeneous coordinate space in which vertex positions (`Position` decoration) are written by `pre-rasterization shader stages`.

Clip Distance

A built-in output from `pre-rasterization shader stages` defining a clip half-space against which the primitive is clipped.

Clip Volume

The intersection of the view volume with all clip half-spaces.

Color Attachment

A subpass attachment point, or image view, that is the target of fragment color outputs and blending.

Color Renderable Format

A `VkFormat` where `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` is set in one of the following, depending on the image's tiling:

- `VkFormatProperties::linearTilingFeatures`
- `VkFormatProperties::optimalTilingFeatures`
- `VkDrmFormatModifierPropertiesEXT::drmFormatModifierTilingFeatures`

Combined Image Sampler

A descriptor type that includes both a sampled image and a sampler.

Command Buffer

An object that records commands to be submitted to a queue. Represented by a `VkCommandBuffer` object.

Command Pool

An object that command buffer memory is allocated from, and that owns that memory.

Command pools aid multithreaded performance by enabling different threads to use different allocators, without internal synchronization on each use. Represented by a [VkCommandPool](#) object.

Compatible Allocator

When allocators are compatible, allocations from each allocator **can** be freed by the other allocator.

Compatible Image Formats

When formats are compatible, images created with one of the formats **can** have image views created from it using any of the compatible formats. Also see *Size-Compatible Image Formats*.

Compatible Queues

Queues within a queue family. Compatible queues have identical properties.

Complete Mipmap Chain

The entire set of mip levels that can be provided for an image, from the largest application-specified mip level size down to the *minimum mip level size*. See [Image Mip Level Sizing](#).

Component (Format)

A distinct part of a format. Color components are represented with **R**, **G**, **B**, and **A**. Depth and stencil components are represented with **D** and **S**. Formats **can** have multiple instances of the same component. Some formats have other notations such as **E** or **X** which are not considered a component of the format.

Compressed Texel Block

An element of an image having a block-compressed format, comprising a rectangular block of texel values that are encoded as a single value in memory. Compressed texel blocks of a particular block-compressed format have a corresponding width, height, and depth defining the dimensions of these elements in units of texels, and a size in bytes of the encoding in memory.

Constant Integral Expressions

A SPIR-V constant instruction whose type is **OpTypeInt**. See *Constant Instruction* in section 2.2.1 “Instructions” of the [Khronos SPIR-V Specification](#).

Coverage Index

The index of a sample in the coverage mask.

Coverage Mask

A bitfield associated with a fragment representing the samples that were determined to be covered based on the result of rasterization, and then subsequently modified by fragment operations or the fragment shader.

Cull Distance

A built-in output from [pre-rasterization shader stages](#) defining a cull half-space where the primitive is rejected if all vertices have a negative value for the same cull distance.

Cull Volume

The intersection of the view volume with all cull half-spaces.

Decoration (SPIR-V)

Auxiliary information such as built-in variables, stream numbers, invariance, interpolation type, relaxed precision, etc., added to variables or structure-type members through decorations.

Depth/Stencil Attachment

A subpass attachment point, or image view, that is the target of depth and/or stencil test operations and writes.

Depth/Stencil Format

A [VkFormat](#) that includes depth and/or stencil components.

Depth/Stencil Image (or ImageView)

A [VkImage](#) (or [VkImageView](#)) with a depth/stencil format.

Depth/Stencil Resolve Attachment

A subpass attachment point, or image view, that is the target of a multisample resolve operation from the corresponding depth/stencil attachment at the end of the subpass.

Derivative Group

A set of fragment shader invocations that cooperate to compute derivatives, including implicit derivatives for sampled image operations.

Descriptor

Information about a resource or resource view written into a descriptor set that is used to access the resource or view from a shader.

Descriptor Binding

An entry in a descriptor set layout corresponding to zero or more descriptors of a single descriptor type in a set. Defined by a [VkDescriptorSetLayoutBinding](#) structure.

Descriptor Pool

An object that descriptor sets are allocated from, and that owns the storage of those descriptor sets. Descriptor pools aid multithreaded performance by enabling different threads to use different allocators, without internal synchronization on each use. Represented by a [VkDescriptorPool](#) object.

Descriptor Set

An object that resource descriptors are written into via the API, and that **can** be bound to a command buffer such that the descriptors contained within it **can** be accessed from shaders. Represented by a [VkDescriptorSet](#) object.

Descriptor Set Layout

An object defining the set of resources (types and counts) and their relative arrangement (in the binding namespace) within a descriptor set. Used when allocating descriptor sets and when creating pipeline layouts. Represented by a [VkDescriptorSetLayout](#) object.

Device

The processor(s) and execution environment that perform tasks requested by the application via the Vulkan API.

Device Group

A set of physical devices that support accessing each other's memory and recording a single command buffer that **can** be executed on all the physical devices.

Device Index

A zero-based integer that identifies one physical device from a logical device. A device index is valid if it is less than the number of physical devices in the logical device.

Device Mask

A bitmask where each bit represents one device index. A device mask value is valid if every bit that is set in the mask is at a bit position that is less than the number of physical devices in the logical device.

Device Memory

Memory accessible to the device. Represented by a [VkDeviceMemory](#) object.

Device-Level Command

Any command that is dispatched from a logical device, or from a child object of a logical device.

Device-Level Functionality

All device-level commands and objects, and their structures, enumerated types, and enumerants. Additionally, physical-device-level functionality defined by a [device extension](#) is also considered device-level functionality.

Device-Level Object

Logical device objects and their child objects. For example, [VkDevice](#), [VkQueue](#), and [VkCommandBuffer](#) objects are device-level objects.

Device-Local Memory

Memory that is connected to the device, and **may** be more performant for device access than host-local memory.

Direct Drawing Commands

Drawing commands that take all their parameters as direct arguments to the command (and not sourced via structures in buffer memory as the *indirect drawing commands*). Includes [vkCmdDraw](#), and [vkCmdDrawIndexed](#).

Disjoint

Disjoint planes are *image planes* to which memory is bound independently.

A *disjoint image* consists of multiple *disjoint planes*, and is created with the [VK_IMAGE_CREATE_DISJOINT_BIT](#) bit set.

Dispatchable Command

A non-global command. The first argument to each dispatchable command is a dispatchable

handle type.

Dispatchable Handle

A handle of a pointer handle type which **may** be used by layers as part of intercepting API commands.

Dispatching Commands

Commands that provoke work using a compute pipeline, such as [vkCmdDispatch](#). See [Dispatching Commands](#).

Drawing Commands

Commands that provoke work using a graphics pipeline. Includes [vkCmdDraw](#), [vkCmdDrawIndexed](#), [vkCmdDrawIndirectCount](#), [vkCmdDrawIndexedIndirectCount](#), [vkCmdDrawIndirect](#), and [vkCmdDrawIndexedIndirect](#).

Duration (Command)

The *duration* of a Vulkan command refers to the interval between calling the command and its return to the caller.

Dynamic Storage Buffer

A storage buffer whose offset is specified each time the storage buffer is bound to a command buffer via a descriptor set.

Dynamic Uniform Buffer

A uniform buffer whose offset is specified each time the uniform buffer is bound to a command buffer via a descriptor set.

Dynamically Uniform

See *Dynamically Uniform* in section 2.2 “Terms” of the [Khronos SPIR-V Specification](#).

Element

Arrays are composed of multiple elements, where each element exists at a unique index within that array. Used primarily to describe data passed to or returned from the Vulkan API.

Explicitly-Enabled Layer

A layer enabled by the application by adding it to the enabled layer list in [vkCreateInstance](#) or [vkCreateDevice](#).

Event

A synchronization primitive that is signaled when execution of previous commands completes through a specified set of pipeline stages. Events can be waited on by the device and polled by the host. Represented by a [VkEvent](#) object.

Executable State (Command Buffer)

A command buffer that has ended recording commands and **can** be executed. See also Initial State and Recording State.

Execution Dependency

A dependency that guarantees that certain pipeline stages' work for a first set of commands has completed execution before certain pipeline stages' work for a second set of commands begins execution. This is accomplished via pipeline barriers, subpass dependencies, events, or implicit ordering operations.

Execution Dependency Chain

A sequence of execution dependencies that transitively act as a single execution dependency.

Explicit chroma reconstruction

An implementation of sampler $Y'C_B C_R$ conversion which reconstructs reduced-resolution chroma samples to luma resolution and then separately performs texture sample interpolation. This is distinct from an implicit implementation, which incorporates chroma sample reconstruction into texture sample interpolation.

Extension Scope

The set of objects and commands that **can** be affected by an extension. Extensions are either device scope or instance scope.

Extending Structure

A structure type which may appear in the *pNext chain* of another structure, extending the functionality of the other structure. Extending structures may be defined by either core API versions or extensions.

External Handle

A resource handle which has meaning outside of a specific Vulkan device or its parent instance. External handles **may** be used to share resources between multiple Vulkan devices in different instances, or between Vulkan and other APIs. Some external handle types correspond to platform-defined handles, in which case the resource **may** outlive any particular Vulkan device or instance and **may** be transferred between processes, or otherwise manipulated via functionality defined by the platform for that handle type.

External synchronization

A type of synchronization **required** of the application, where parameters defined to be externally synchronized **must** not be used simultaneously in multiple threads.

Facingness (Polygon)

A classification of a polygon as either front-facing or back-facing, depending on the orientation (winding order) of its vertices.

Facingness (Fragment)

A fragment is either front-facing or back-facing, depending on the primitive it was generated from. If the primitive was a polygon (regardless of polygon mode), the fragment inherits the facingness of the polygon. All other fragments are front-facing.

Fence

A synchronization primitive that is signaled when a set of batches or sparse binding operations complete execution on a queue. Fences **can** be waited on by the host. Represented by a [VkFence](#)

object.

Flat Shading

A property of a vertex attribute that causes the value from a single vertex (the provoking vertex) to be used for all vertices in a primitive, and for interpolation of that attribute to return that single value unaltered.

Format Features

A set of features from [VkFormatFeatureFlagBits](#) that a [VkFormat](#) is capable of using for various commands. The list is determined by factors such as [VkImageTiling](#).

Fragment

A rectangular framebuffer region with associated data produced by [rasterization](#) and processed by [fragment operations](#) including the fragment shader.

Fragment Area

The width and height, in pixels, of a fragment.

Fragment Input Attachment Interface

Variables with [UniformConstant](#) storage class and a decoration of [InputAttachmentIndex](#) that are statically used by a fragment shader's entry point, which receive values from input attachments.

Fragment Output Interface

A fragment shader entry point's variables with [Output](#) storage class, which output to color and/or depth/stencil attachments.

Framebuffer

A collection of image views and a set of dimensions that, in conjunction with a render pass, define the inputs and outputs used by drawing commands. Represented by a [VkFramebuffer](#) object.

Framebuffer Attachment

One of the image views used in a framebuffer.

Framebuffer Coordinates

A coordinate system in which adjacent pixels' coordinates differ by 1 in x and/or y, with (0,0) in the upper left corner and pixel centers at half-integers.

Framebuffer-Space

Operating with respect to framebuffer coordinates.

Framebuffer-Local

A framebuffer-local dependency guarantees that only for a single framebuffer region, the first set of operations happens-before the second set of operations.

Framebuffer-Global

A framebuffer-global dependency guarantees that for all framebuffer regions, the first set of operations happens-before the second set of operations.

Framebuffer Region

A framebuffer region is a set of sample (x, y, layer, sample) coordinates that is a subset of the entire framebuffer.

Front-Facing

See Facingness.

Full Compatibility

A given version of the API is fully compatible with another version if an application, relying only on valid behavior and functionality defined by either of those specifications, is able to correctly run against each version without any modification. This assumes no active attempt by that application to not run when it detects a different version.

Global Command

A Vulkan command for which the first argument is not a dispatchable handle type.

Global Workgroup

A collection of local workgroups dispatched by a single dispatching command.

Handle

An opaque integer or pointer value used to refer to a Vulkan object. Each object type has a unique handle type.

Happen-after, happens-after

A transitive, irreflexive and antisymmetric ordering relation between operations. An execution dependency with a source of **A** and a destination of **B** enforces that **B** happens-after **A**. The inverse relation of happens-before.

Happen-before, happens-before

A transitive, irreflexive and antisymmetric ordering relation between operations. An execution dependency with a source of **A** and a destination of **B** enforces that **A** happens-before **B**. The inverse relation of happens-after.

Helper Invocation

A fragment shader invocation that is created solely for the purposes of evaluating derivatives for use in non-helper fragment shader invocations, and which does not have side effects.

Host

The processor(s) and execution environment that the application runs on, and that the Vulkan API is exposed on.

Host Mapped Device Memory

Device memory that is mapped for host access using [vkMapMemory](#).

Host Mapped Foreign Memory

Memory owned by a foreign device that is mapped for host access.

Host Memory

Memory not accessible to the device, used to store implementation data structures.

Host-Accessible Subresource

A buffer, or a linear image subresource in either the `VK_IMAGE_LAYOUT_PREINITIALIZED` or `VK_IMAGE_LAYOUT_GENERAL` layout. Host-accessible subresources have a well-defined addressing scheme which can be used by the host.

Host-Local Memory

Memory that is not local to the device, and **may** be less performant for device access than device-local memory.

Host-Visible Memory

Device memory that **can** be mapped on the host and **can** be read and written by the host.

ICD

Installable Client Driver. An ICD is represented as a `VkPhysicalDevice`.

Identically Defined Objects

Objects of the same type where all arguments to their creation or allocation functions, with the exception of `pAllocator`, are

1. Vulkan handles which refer to the same object or
2. identical scalar or enumeration values or
3. Host pointers which point to an array of values or structures which also satisfy these three constraints.

Image

A resource that represents a multi-dimensional formatted interpretation of device memory. Represented by a `VkImage` object.

Image Subresource

A specific mipmap level, layer, and set of aspects of an image.

Image Subresource Range

A set of image subresources that are contiguous mipmap levels and layers.

Image View

An object that represents an image subresource range of a specific image, and state controlling how the contents are interpreted. Represented by a `VkImageView` object.

Immutable Sampler

A sampler descriptor provided at descriptor set layout creation time for a specific binding. This sampler is then used for that binding in all descriptor sets allocated with the layout, and it **cannot** be changed.

Implicit chroma reconstruction

An implementation of sampler $Y'C_B C_R$ conversion which reconstructs the reduced-resolution

chroma samples directly at the sample point, as part of the normal texture sampling operation. This is distinct from an *explicit chroma reconstruction* implementation, which reconstructs the reduced-resolution chroma samples to the resolution of the luma samples, then filters the result as part of texture sample interpolation.

Implicitly-Enabled Layer

A layer enabled by a loader-defined mechanism outside the Vulkan API, rather than explicitly by the application during instance or device creation.

Index Buffer

A buffer bound via [vkCmdBindIndexBuffer](#) which is the source of index values used to fetch vertex attributes for a [vkCmdDrawIndexed](#) or [vkCmdDrawIndexedIndirect](#) command.

Indexed Drawing Commands

Drawing commands which use an *index buffer* as the source of index values used to fetch vertex attributes for a drawing command. Includes [vkCmdDrawIndexed](#), [vkCmdDrawIndexedIndirectCount](#), and [vkCmdDrawIndexedIndirect](#).

Indirect Commands

Drawing or dispatching commands that source some of their parameters from structures in buffer memory. Includes [vkCmdDrawIndirect](#), [vkCmdDrawIndexedIndirect](#), [vkCmdDrawIndirectCount](#), [vkCmdDrawIndexedIndirectCount](#), and [vkCmdDispatchIndirect](#).

Indirect Drawing Commands

Drawing commands that source some of their parameters from structures in buffer memory. Includes [vkCmdDrawIndirect](#), [vkCmdDrawIndirectCount](#), [vkCmdDrawIndexedIndirectCount](#), and [vkCmdDrawIndexedIndirect](#).

Initial State (Command Buffer)

A command buffer that has not begun recording commands. See also Recording State and Executable State.

Input Attachment

A descriptor type that represents an image view, and supports unfiltered read-only access in a shader, only at the fragment's location in the view.

Instance

The top-level Vulkan object, which represents the application's connection to the implementation. Represented by a [VkInstance](#) object.

Instance-Level Command

Any command that is dispatched from an instance, or from a child object of an instance, except for physical devices and their children.

Instance-Level Functionality

All instance-level commands and objects, and their structures, enumerated types, and enumerants.

Instance-Level Object

High-level Vulkan objects, which are not physical devices, nor children of physical devices. For example, [VkInstance](#) is an instance-level object.

Instance (Memory)

In a logical device representing more than one physical device, some device memory allocations have the requested amount of memory allocated multiple times, once for each physical device in a device mask. Each such replicated allocation is an instance of the device memory.

Instance (Resource)

In a logical device representing more than one physical device, buffer and image resources exist on all physical devices but **can** be bound to memory differently on each. Each such replicated resource is an instance of the resource.

Internal Synchronization

A type of synchronization **required** of the implementation, where parameters not defined to be externally synchronized **may** require internal mutexing to avoid multithreaded race conditions.

Invocation (Shader)

A single execution of an entry point in a SPIR-V module. For example, a single vertex's execution of a vertex shader or a single fragment's execution of a fragment shader.

Invocation Group

A set of shader invocations that are executed in parallel and that **must** execute the same control flow path in order for control flow to be considered dynamically uniform.

Legacy (feature)

A feature which is no longer recommended as the correct or best way to achieve its intended purpose. Legacy features are typically superseded by new features which are the currently recommended way to achieve that purpose.

Linear Resource

A resource is *linear* if it is one of the following:

- a [VkBuffer](#)
- a [VkImage](#) created with `VK_IMAGE_TILING_LINEAR`
- a [VkImage](#) created with `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` and whose [Linux DRM format modifier](#) is `DRM_FORMAT_MOD_LINEAR`

A resource is *non-linear* if it is one of the following:

- a [VkImage](#) created with `VK_IMAGE_TILING_OPTIMAL`
- a [VkImage](#) created with `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` and whose [Linux DRM format modifier](#) is not `DRM_FORMAT_MOD_LINEAR`

Linux DRM Format Modifier

A 64-bit, vendor-prefixed, semi-opaque unsigned integer describing vendor-specific details of an

image's memory layout. In Linux graphics APIs, *modifiers* are commonly used to specify the memory layout of externally shared images. An image has a *modifier* if and only if it is created with *tiling* equal to `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`. For more details, refer to the appendix for extension `VK_EXT_image_drm_format_modifier`.

Local Workgroup

A collection of compute shader invocations invoked by a single dispatching command, which share data via `WorkgroupLocal` variables and can synchronize with each other.

Logical Device

An object that represents the application's interface to the physical device. The logical device is the parent of most Vulkan objects. Represented by a `VkDevice` object.

Logical Operation

Bitwise operations between a fragment color value and a value in a color attachment, that produce a final color value to be written to the attachment.

Lost Device

A state that a logical device **may** be in as a result of unrecoverable implementation errors, or other exceptional conditions.

Mappable

See Host-Visible Memory.

Memory Dependency

A memory dependency is an execution dependency which includes availability and visibility operations such that:

- The first set of operations happens-before the availability operation
- The availability operation happens-before the visibility operation
- The visibility operation happens-before the second set of operations

Memory Domain

A memory domain is an abstract place to which memory writes are made available by availability operations and memory domain operations. The memory domains correspond to the set of agents that the write **can** then be made visible to. The memory domains are *host*, *device*, *shader*, *workgroup instance* (for workgroup instance there is a unique domain for each compute workgroup) and *subgroup instance* (for subgroup instance there is a unique domain for each subgroup).

Memory Domain Operation

An operation that makes the writes that are available to one memory domain available to another memory domain.

Memory Heap

A region of memory from which device memory allocations **can** be made.

Memory Type

An index used to select a set of memory properties (e.g. mappable, cached) for a device memory allocation.

Minimum Mip Level Size

The smallest size that is permitted for a mip level. For conventional images this is 1x1x1. See [Image Mip Level Sizing](#).

Mip Tail Region

The set of mipmap levels of a sparse residency texture that are too small to fill a sparse block, and that **must** all be bound to memory collectively and opaquely.

Multi-planar

A *multi-planar format* (or “planar format”) is an image format consisting of more than one *plane*, identifiable with a `_2PLANE` or `_3PLANE` component to the format name and listed in [Formats Requiring Sampler Y'C_BC_R Conversion for VK_IMAGE_ASPECT_COLOR_BIT Image Views](#). A *multi-planar image* (or “planar image”) is an image of a multi-planar format.

Non-Dispatchable Handle

A handle of an integer handle type. Handle values **may** not be unique, even for two objects of the same type.

Non-Indexed Drawing Commands

Drawing commands for which the vertex attributes are sourced in linear order from the vertex input attributes for a drawing command (i.e. they do not use an *index buffer*). Includes [vkCmdDraw](#), [vkCmdDrawIndirectCount](#), and [vkCmdDrawIndirect](#).

Normalized

A value that is interpreted as being in the range [0,1] as a result of being implicitly divided by some other value.

Normalized Device Coordinates

A coordinate space after perspective division is applied to clip coordinates, and before the viewport transformation converts them to framebuffer coordinates.

Obsoleted (feature)

A feature is obsolete if it can no longer be used.

Opaque Capture Address

A 64-bit value representing the device address of a buffer or memory object that is expected to be used by trace capture/replay tools in combination with the `bufferDeviceAddress` feature.

Overlapped Range (Aliased Range)

The aliased range of a device memory allocation that intersects a given image subresource of an image or range of a buffer.

Ownership (Resource)

If an entity (e.g. a queue family) has ownership of a resource, access to that resource is well-

defined for access by that entity.

Packed Format

A format whose components are stored as a single texel block in memory, with their relative locations defined within that element.

Payload

Importable or exportable reference to the internal data of an object in Vulkan.

Peer Memory

An instance of memory corresponding to a different physical device than the physical device performing the memory access, in a logical device that represents multiple physical devices.

Physical Device

An object that represents a single device in the system. Represented by a [VkPhysicalDevice](#) object.

Physical-Device-Level Command

Any command that is dispatched from a physical device.

Physical-Device-Level Functionality

All physical-device-level commands and objects, and their structures, enumerated types, and enumerants.

Physical-Device-Level Object

Physical device objects. For example, [VkPhysicalDevice](#) is a physical-device-level object.

Pipeline

An object controlling how graphics or compute work is executed on the device. A pipeline includes one or more shaders, as well as state controlling any non-programmable stages of the pipeline. Represented by a [VkPipeline](#) object.

Pipeline Barrier

An execution and/or memory dependency recorded as an explicit command in a command buffer, that forms a dependency between the previous and subsequent commands.

Pipeline Cache

An object that **can** be used to collect and retrieve information from pipelines as they are created, and **can** be populated with previously retrieved information in order to accelerate pipeline creation. Represented by a [VkPipelineCache](#) object.

Pipeline JSON Schema

A JSON-based representation for encapsulating all pipeline state which is necessary for the offline pipeline cache compiler. This includes the SPIR-V shader module, pipeline layout, render pass information and pipeline state creation information.

Pipeline Layout

An object defining the set of resources (via a collection of descriptor set layouts) and push

constants used by pipelines that are created using the layout. Used when creating a pipeline and when binding descriptor sets and setting push constant values. Represented by a [VkPipelineLayout](#) object.

Pipeline Stage

A logically independent execution unit that performs some of the operations defined by an action command.

Pipeline Identifier

An identifier that can be used to identify a specific pipeline independently from the pipeline description.

pNext Chain

A set of structures [chained together](#) through their [pNext](#) members.

Planar

See *multi-planar*.

Plane

An *image plane* is part of the representation of an image, containing a subset of the color components necessary to represent the texels in the image and with a contiguous mapping of coordinates to bound memory. Most images consist only of a single plane, but some formats spread the components across multiple image planes. The host-accessible properties of each image plane are accessible for a linear layout using [vkGetImageSubresourceLayout](#). If a multi-planar image is created with the [VK_IMAGE_CREATE_DISJOINT_BIT](#) bit set, the image is described as *disjoint*, and its planes are therefore bound to memory independently.

Point Sampling (Rasterization)

A rule that determines whether a fragment sample location is covered by a polygon primitive by testing whether the sample location is in the interior of the polygon in framebuffer-space, or on the boundary of the polygon according to the tie-breaking rules.

Potential Format Features

The union of all [VkFormatFeatureFlagBits](#) that the implementation supports for a specified [VkFormat](#), over all supported image tilings. For [QNX Screen external formats](#) the [VkFormatFeatureFlagBits](#) is provided by the implementation.

Pre-rasterization

Operations that execute before [rasterization](#), and any state associated with those operations.

Presentable image

A [VkImage](#) object obtained from a [VkSwapchainKHR](#) used to present to a [VkSurfaceKHR](#) object.

Preserve Attachment

One of a list of attachments in a subpass description that is not read or written by the subpass, but that is read or written on earlier and later subpasses and whose contents **must** be preserved through this subpass.

Primary Command Buffer

A command buffer that **can** execute secondary command buffers, and **can** be submitted directly to a queue.

Primitive Topology

State controlling how vertices are assembled into primitives, e.g. as lists of triangles, strips of lines, etc.

Promoted (feature)

A feature from an older extension is considered promoted if it is made available as part of a new core version or newer extension with wider support.

Protected Buffer

A buffer to which protected device memory **can** be bound.

Protected-capable Device Queue

A device queue to which protected command buffers **can** be submitted.

Protected Command Buffer

A command buffer which **can** be submitted to a protected-capable device queue.

Protected Device Memory

Device memory which **can** be visible to the device but **must** not be visible to the host.

Protected Image

An image to which protected device memory **can** be bound.

Provisional

A feature is released provisionally in order to get wider feedback on the functionality before it is finalized. Provisional features may change in ways that break backwards compatibility, and thus are not recommended for use in production applications.

Provoking Vertex

The vertex in a primitive from which flat shaded attribute values are taken. This is generally the “first” vertex in the primitive, and depends on the primitive topology.

Push Constants

A small bank of values writable via the API and accessible in shaders. Push constants allow the application to set values used in shaders without creating buffers or modifying and binding descriptor sets for each update.

Push Constant Interface

The set of variables with **PushConstant** storage class that are statically used by a shader entry point, and which receive values from push constant commands.

Descriptor Update Template

An object specifying a mapping from descriptor update information in host memory to elements in a descriptor set, which helps enable more efficient descriptor set updates.

Query Pool

An object containing a number of query entries and their associated state and results. Represented by a [VkQueryPool](#) object.

Queue

An object that executes command buffers and sparse binding operations on a device. Represented by a [VkQueue](#) object.

Queue Family

A set of queues that have common properties and support the same functionality, as advertised in [VkQueueFamilyProperties](#).

Queue Operation

A unit of work to be executed by a specific queue on a device, submitted via a [queue submission command](#). Each queue submission command details the specific queue operations that occur as a result of calling that command. Queue operations typically include work that is specific to each command, and synchronization tasks.

Queue Submission

Zero or more batches and an optional fence to be signaled, passed to a command for execution on a queue. See the [Devices and Queues chapter](#) for more information.

Recording State (Command Buffer)

A command buffer that is ready to record commands. See also Initial State and Executable State.

Release Operation (Resource)

An operation that releases ownership of an image subresource or buffer range.

Render Pass

An object that represents a set of framebuffer attachments and phases of rendering using those attachments. Represented by a [VkRenderPass](#) object.

Render Pass Instance

A use of a render pass in a command buffer.

Required Extensions

Extensions that **must** be enabled alongside extensions dependent on them (see [Extension Dependencies](#)).

Reset (Command Buffer)

Resetting a command buffer discards any previously recorded commands and puts a command buffer in the initial state.

Residency Code

An integer value returned by sparse image instructions, indicating whether any sparse unbound texels were accessed.

Resolve Attachment

A subpass attachment point, or image view, that is the target of a multisample resolve operation from the corresponding color attachment at the end of the subpass.

Retired Swapchain

A swapchain that has been used as the `oldSwapchain` parameter to `vkCreateSwapchainKHR`. Images cannot be acquired from a retired swapchain, however images that were acquired (but not presented) before the swapchain was retired **can** be presented.

Sample Index

The index of a sample within a [single set of samples](#).

Sample Shading

Invoking the fragment shader multiple times per fragment, with the covered samples partitioned among the invocations.

Sampled Image

A descriptor type that represents an image view, and supports filtered (sampled) and unfiltered read-only access in a shader.

Sampler

An object containing state controlling how sampled image data is sampled (or filtered) when accessed in a shader. Also a descriptor type describing the object. Represented by a [VkSampler](#) object.

Secondary Command Buffer

A command buffer that **can** be executed by a primary command buffer, and **must** not be submitted directly to a queue.

Self-Dependency

A subpass dependency from a subpass to itself, i.e. with `srcSubpass` equal to `dstSubpass`. A self-dependency is not automatically performed during a render pass instance, rather a subset of it **can** be performed via [vkCmdPipelineBarrier](#) during the subpass.

Semaphore

A synchronization primitive that supports signal and wait operations, and **can** be used to synchronize operations within a queue or across queues. Represented by a [VkSemaphore](#) object.

Shader

Instructions selected (via an entry point) from a shader module, which are executed in a shader stage.

Shader Code

A stream of instructions used to describe the operation of a shader.

Shader Module

A collection of shader code, potentially including several functions and entry points, that is used to create shaders in pipelines. Represented by a [VkShaderModule](#) object.

Shader Stage

A stage of the graphics or compute pipeline that executes shader code.

Shading Rate

The ratio of the number of fragment shader invocations generated in a fully covered framebuffer region to the size (in pixels) of that region.

Shared presentable image

A presentable image created from a swapchain with `VkPresentModeKHR` set to either `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` or `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`.

Side Effect

A store to memory or atomic operation on memory from a shader invocation.

Single Event Upset

A change of physical device state, such as a register or memory bitflip, e.g. caused by ionizing radiation.

Single-plane format

A format that is not *multi-planar*.

Size-Compatible Image Formats

When a compressed image format and an uncompressed image format are size-compatible, it means that the texel block size of the uncompressed format **must** equal the texel block size of the compressed format.

Sparse Block

An element of a sparse resource that can be independently bound to memory. Sparse blocks of a particular sparse resource have a corresponding size in bytes that they use in the bound memory.

Sparse Image Block

A sparse block in a sparse partially-resident image. In addition to the sparse block size in bytes, sparse image blocks have a corresponding width, height, and depth defining the dimensions of these elements in units of texels or compressed texel blocks, the latter being used in case of sparse images having a block-compressed format.

Sparse Unbound Texel

A texel read from a region of a sparse texture that does not have memory bound to it.

Static Use

An object in a shader is statically used by a shader entry point if any function in the entry point's call tree contains an instruction using the object. A reference in the entry point's interface list does not constitute a static use. Static use is used to constrain the set of descriptors used by a shader entry point.

Storage Buffer

A descriptor type that represents a buffer, and supports reads, writes, and atomics in a shader.

Storage Image

A descriptor type that represents an image view, and supports unfiltered loads, stores, and atomics in a shader.

Storage Texel Buffer

A descriptor type that represents a buffer view, and supports unfiltered, formatted reads, writes, and atomics in a shader.

Subgroup

A set of shader invocations that **can** synchronize and share data with each other efficiently. In compute shaders, the *local workgroup* is a superset of the subgroup.

Subgroup Mask

A bitmask for all invocations in the current subgroup with one bit per invocation, starting with the least significant bit in the first vector component, continuing to the last bit (less than `SubgroupSize`) in the last required vector component.

Subpass

A phase of rendering within a render pass, that reads and writes a subset of the attachments.

Subpass Dependency

An execution and/or memory dependency between two subpasses described as part of render pass creation, and automatically performed between subpasses in a render pass instance. A subpass dependency limits the overlap of execution of the pair of subpasses, and **can** provide guarantees of memory coherence between accesses in the subpasses.

Subpass Description

Lists of attachment indices for input attachments, color attachments, depth/stencil attachment, resolve attachments, depth/stencil resolve, and preserve attachments used by the subpass in a render pass.

Subset (Self-Dependency)

A subset of a self-dependency is a pipeline barrier performed during the subpass of the self-dependency, and whose stage masks and access masks each contain a subset of the bits set in the identically named mask in the self-dependency.

Superseded (feature)

A legacy feature can be superseded by newer functionality that achieves the same goals through a newer style of API, being more efficient, or otherwise following current recommendations for use of Vulkan.

Texel Block

A single addressable element of an image with an uncompressed [VkFormat](#), or a single compressed block of an image with a compressed [VkFormat](#).

Texel Block Size

The size (in bytes) used to store a texel block of a compressed or uncompressed image.

Texel Coordinate System

One of three coordinate systems (normalized, unnormalized, integer) defining how texel coordinates are interpreted in an image or a specific mipmap level of an image.

Timeline Semaphore

A semaphore with a strictly increasing 64-bit unsigned integer payload indicating whether the semaphore is signaled with respect to a particular reference value. Represented by a [VkSemaphore](#) object created with a semaphore type of `VK_SEMAPHORE_TYPE_TIMELINE`.

Uniform Texel Buffer

A descriptor type that represents a buffer view, and supports unfiltered, formatted, read-only access in a shader.

Uniform Buffer

A descriptor type that represents a buffer, and supports read-only access in a shader.

Units in the Last Place (ULP)

A measure of floating-point error loosely defined as the smallest representable step in a floating-point format near a given value. For the precise definition see [Precision and Operation of SPIR-V instructions](#) or Jean-Michel Muller, “On the definition of ulp(x)”, RR-5504, INRIA. Other sources may also use the term “unit of least precision”.

Unnormalized

A value that is interpreted according to its conventional interpretation, and is not normalized.

Unprotected Buffer

A buffer to which unprotected device memory **can** be bound.

Unprotected Command Buffer

A command buffer which **can** be submitted to an unprotected device queue or a protected-capable device queue.

Unprotected Device Memory

Device memory which **can** be visible to the device and **can** be visible to the host.

Unprotected Image

An image to which unprotected device memory **can** be bound.

User-Defined Variable Interface

A shader entry point’s variables with `Input` or `Output` storage class that are not built-in variables.

Vertex Input Attribute

A graphics pipeline resource that produces input values for the vertex shader by reading data from a vertex input binding and converting it to the attribute’s format.

Variable-Sized Descriptor Binding

A descriptor binding whose size will be specified when a descriptor set is allocated using this layout.

Vertex Input Binding

A graphics pipeline resource that is bound to a buffer and includes state that affects addressing calculations within that buffer.

Vertex Input Interface

A vertex shader entry point's variables with **Input** storage class, which receive values from vertex input attributes.

View Mask

When multiview is enabled, a view mask is a property of a subpass controlling which views the rendering commands are broadcast to.

View Volume

A subspace in homogeneous coordinates, corresponding to post-projection x and y values between -1 and +1, and z values between 0 and +1.

Viewport Transformation

A transformation from normalized device coordinates to framebuffer coordinates, based on a viewport rectangle and depth range.

Visibility Operation

An operation that causes available values to become visible to specified memory accesses.

Visible

A state of values written to memory that allows them to be accessed by a set of operations.

Common Abbreviations

The abbreviations and acronyms defined in this section are sometimes used in the Specification and the API where they are considered clear and commonplace.

Src

Source

Dst

Destination

Min

Minimum

Max

Maximum

Rect

Rectangle

Info

Information

LOD

Level of Detail

Log

Logarithm

ID

Identifier

UUID

Universally Unique Identifier

Op

Operation

R

Red color component

G

Green color component

B

Blue color component

A

Alpha color component

RTZ

Round towards zero

RTE

Round to nearest even

Prefixes

Prefixes are used in the API to denote specific semantic meaning of Vulkan names, or as a label to avoid name clashes, and are explained here:

VK/Vk/vk

Vulkan namespace

All types, commands, enumerants and defines in this specification are prefixed with these two characters.

PFN/pfn

Function Pointer

Denotes that a type is a function pointer, or that a variable is of a pointer type.

p

Pointer

Variable is a pointer.

vkCmd

Commands that record commands in command buffers

These API commands do not result in immediate processing on the device. Instead, they record the requested action in a command buffer for execution when the command buffer is submitted to a queue.

s

Structure

Used to denote the `VK_STRUCTURE_TYPE*` member of each structure in `sType`

Appendix K: Credits (Informative)

Vulkan SC 1.0 is the result of contributions from many people and companies participating in the Khronos Vulkan SC Working Group, building upon the Base Vulkan specification produced by the Khronos Vulkan Working Group, as well as input from the Vulkan Advisory Panel.

Members of the Working Group, including the company that they represented at the time of their most recent contribution, are listed in the following sections. Some specific contributions made by individuals are listed together with their name.

Working Group Contributors to Vulkan SC 1.0

- Aarusha Thakral, CoreAVI
- Aidan Fabius, CoreAVI
- Alastair Donaldson, Google
- Alastair Murray, Codeplay Software Ltd.
- Alex Crabb, Khronos
- Alexander Galazin, Arm
- Alis Ors, NXP Semiconductors
- Alon Or-bach, Samsung Electronics
- Andrew Garrard, Imagination Technologies
- Anna Buczkowska, Mobica
- Balajee Gurumoorah, Huawei Technologies Co., Ltd.
- Bogdan Naodovic, NVIDIA
- Boris Zanin, Mobica
- Brad Cain, NVIDIA
- Cary Ashby, Collins Aerospace
- Chris Forbes, Google
- Craig Davies, Huawei Technologies Co., Ltd.
- Daniel Bernal, Arm
- Daniel Koch, NVIDIA
- Dave Higham, Imagination Technologies
- Dave McCloskey, Juice Labs
- David Hayward, Imagination Technologies
- Donald Scorgie, Imagination Technologies
- Doug Singkofer, Collins Aerospace
- Emily Stearns, Khronos

- Erik Tomusk, Codeplay Software Ltd.
- Ewa Galamon, Mobica
- Greg Szober, CoreAVI
- Illya Rudkin, Codeplay Software Ltd.
- Jacek Wisniewski, Mobica
- James Helferty, NVIDIA
- Jan Hemes, Continental Corporation
- Jan-Harald Fredriksen, Arm
- Janos Lakatos, Imagination Technologies
- Jeff Bolz, NVIDIA
- Jim Carroll, Mobica
- John Zulauf, LunarG
- Jon Leech, Independent (XML toolchain, normative language, release wrangler)
- Jun Wang, Huawei Technologies Co., Ltd.
- Karen Ghavam, LunarG
- Karolina Palka, Mobica
- Ken Wenger, CoreAVI
- Lenny Komow, LunarG
- Lilja Tamminen, Basemark Oy
- Luca Di Mauro, Arm
- Lukasz Janyst, Daedalean
- Mark Bellamy, Arm
- Matthew Netsch, Qualcomm Technologies, Inc.
- Michael Wong, Codeplay Software Ltd.
- Mukund Keshava, NVIDIA
- Neil Stroud, CoreAVI
- Neil Trevett, NVIDIA
- Nick Blurton-Jones, CoreAVI
- Pawel Ksiezopolski, Mobica
- Piotr Byszewski, Mobica
- Rob Simpson, Qualcomm Technologies, Inc.
- Stephne Strahn, Kalray
- Steve Viggers, CoreAVI (working group chair)
- Tim Lewis, Khronos
- Todd Brown, Collins Aerospace

- Tom Malnar, CoreAVI
- Tom Olson, Arm
- Tony Zlatinski, NVIDIA
- Vladyslav Zakkarchenko, Huawei Technologies Co., Ltd.

Working Group Contributors to Vulkan

- Aaron Greig, Codeplay Software Ltd. (versions 1.1, 1.4)
- Aaron Hagan, AMD (versions 1.1)
- Abhishek Sashi Nair, Huawei Technologies Co., Ltd. (version 1.4)
- Adam Jackson, Red Hat (versions 1.0, 1.1)
- Adam Śmigielski, Mobica (version 1.0)
- Aditi Verma, Qualcomm Technologies, Inc. (version 1.3)
- Ahmed Abdelkhalek, AMD (versions 1.3, 1.4) (video subgroup chair)
- Aidan Fabius, Core Avionics & Industrial Inc. (version 1.2)
- Aitor Camacho, LunarG (version 1.4)
- Alan Baker, Google (versions 1.1, 1.2, 1.3, 1.4)
- Alan Vines, Imagination Technologies (version 1.4)
- Alan Ward, Google (versions 1.1, 1.2)
- Alastair Murray, Codeplay (version 1.4)
- Alejandro Piñeiro, Igalia (version 1.1)
- Aleks Todorov, Google (version 1.4)
- Aleksander Netzel, Epic Games (version 1.4)
- Alex Bourd, Qualcomm Technologies, Inc. (versions 1.0, 1.1, 1.4)
- Alex Crabb, Caster Communications (versions 1.2, 1.3, 1.4)
- Alex Dai, Oppo (version 1.4)
- Alex Krstic, Qualcomm Technologies, Inc. (version 1.4)
- Alex Todorov, Google (version 1.4)
- Alex Walters, Imagination Technologies (versions 1.2, 1.3, 1.4)
- Alexander Galazin, Imagination Technologies (versions 1.0, 1.1, 1.2, 1.3, 1.4) (conformance test subgroup chair)
- Alexey Knyazev, Independent, Khronos (version 1.4)
- Alexey Sachkov, Intel (version 1.3)
- Ali Asif, Huawei Technologies Co., Ltd. (version 1.4)
- Allan MacKinnon, Google (version 1.3)
- Allen Hux, Intel (version 1.0)

- Alon Or-bach, Arm (versions 1.0, 1.1, 1.2, 1.3, 1.4) (system integration subgroup chair)
- Alyssa Rosenzweig, Valve Software (version 1.4)
- Anastasia Stulova, Arm (versions 1.2, 1.3, 1.4)
- Andreas Vasilakis, Think Silicon (version 1.2)
- Andres Gomez, Igalia (version 1.1)
- Andrew Cox, Samsung Electronics (version 1.0)
- Andrew Ellem, Google (version 1.3)
- Andrew Garrard, Imagination Technologies (versions 1.0, 1.1, 1.2, 1.3, 1.4) (format wrangler)
- Andrew Poole, Samsung Electronics (version 1.0)
- Andrew Rafter, Samsung Electronics (version 1.0)
- Andrew Richards, Codeplay Software Ltd. (version 1.0)
- Andrew Woloszyn, Google (versions 1.0, 1.1)
- Andrew Zhi, AMD (version 1.4)
- Ann Thorsnes, Khronos (versions 1.2, 1.3)
- Anthony Salvi, Adobe (version 1.4)
- Antoine Labour, Google (versions 1.0, 1.1)
- Anton Berko, Mediatek (version 1.4)
- Anuj Phogat, Netflix (version 1.4)
- Aras Prancėvičius, Unity Technologies (version 1.0)
- Arpit Agarwal, Qualcomm Technologies, Inc. (version 1.4)
- Arseny Kapoulkine, Roblox (version 1.3)
- Ashley Smith, AMD (version 1.4)
- Ashwin Kolhe, NVIDIA (version 1.0)
- Aurelio Reis, NVIDIA (version 1.4)
- Avinash Baliga, NVIDIA (version 1.4)
- Balaji Calidas, Qualcomm Technologies, Inc. (version 1.4)
- Baldur Karlsson, Valve Software (versions 1.1, 1.2, 1.3, 1.4)
- Barthold Lichtenbelt, NVIDIA (version 1.1)
- Bas Nieuwenhuizen, Google (versions 1.1, 1.2, 1.4)
- Ben Ashbaugh, Intel (version 1.4)
- Ben Bowman, Imagination Technologies (version 1.0)
- Benj Lipchak, Unknown (version 1.0)
- Bill Hollings, Brenwill (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Bill Licea-Kane, Qualcomm Technologies, Inc. (versions 1.0, 1.1)
- Blaine Kohl, Khronos (versions 1.2, 1.3)

- Bob Fraser, Google (version 1.3)
- Boris Zanin, Mobica (versions 1.2, 1.3)
- Brad Grantham, LunarG (version 1.4)
- Brent E. Insko, Intel (version 1.0)
- Brian Ellis, Qualcomm Technologies, Inc. (version 1.0)
- Brian Paul, VMware (versions 1.2, 1.3)
- CAI Kangying, Huawei Technologies Co., Ltd. (version 1.4)
- Caio Marcelo de Oliveira Filho, Intel (versions 1.2, 1.3)
- Calle Ledjefors, Tencent (version 1.4)
- Cass Everitt, Oculus VR (versions 1.0, 1.1)
- Cemil Azizoglu, Canonical (version 1.0)
- Chang-Hyo Yu, Samsung Electronics (version 1.0)
- Charles Giessen, LunarG (versions 1.3, 1.4)
- Charlie Lao, Google (version 1.4)
- Charlie Turner, Igalia (version 1.4)
- Cheng Ge, Tencent (version 1.4)
- Chengping Luo, Mediatek (version 1.4)
- Chia-I Wu, LunarG (version 1.0)
- Chris Bieneman, Microsoft (version 1.4)
- Chris Frascati, Qualcomm Technologies, Inc. (version 1.0)
- Chris Glover, Google (versions 1.3, 1.4)
- Christian Forfang, Arm (version 1.3)
- Christoph Kubisch, NVIDIA (versions 1.3, 1.4)
- Christophe Riccio, LunarG (versions 1.0, 1.1, 1.4)
- Cody Northrop, LunarG (version 1.0)
- Colin Riley, AMD (version 1.1)
- Connor Abbott, Valve Software (version 1.4)
- Caterina Shablia, Collabora (version 1.4)
- Cort Stratton, Google (versions 1.1, 1.2)
- Courtney Goeltzenleuchter, Google (versions 1.0, 1.1, 1.3)
- Craig Davies, Huawei Technologies Co., Ltd. (version 1.2)
- Craig Graham, Samsung Electronics (version 1.4)
- Dae Kim, Imagination Technologies (versions 1.1, 1.4)
- Damien Leone, NVIDIA (version 1.0)
- Dan Baker, Oxide Games (versions 1.0, 1.1)

- Dan Ginsburg, Valve Software (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Dan Sinclair, Google (version 1.4)
- Daniel Johnston, Intel (versions 1.0, 1.1)
- Daniel Koch, NVIDIA (versions 1.0, 1.1, 1.2, 1.3, 1.4) (ray tracing subgroup chair)
- Daniel Rakos, RasterGrid (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Daniel Schürmann, Valve Software (version 1.4)
- Daniel Stone, Collabora (versions 1.1, 1.2, 1.4)
- Daniel Story, Nintendo (version 1.4)
- Daniel Vetter, Intel (version 1.2)
- Daniele Vettorel, Google (version 1.4)
- Danny Huynh, Samsung Electronics (version 1.4)
- Dave Aronson, Adobe (version 1.4)
- Dave Desormeaux, LunarG (version 1.4)
- Dave Evans, Samsung Electronics (version 1.4)
- David Airlie, Red Hat (versions 1.0, 1.1, 1.2, 1.3)
- David Mao, AMD (versions 1.0, 1.2)
- David McCloskey, Juice Labs (version 1.4)
- David Miller, Miller & Mattson (versions 1.0, 1.1) (Vulkan reference card)
- David Neto, Google (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- David Pankratz, Huawei Technologies Co., Ltd. (version 1.3)
- David Wilkinson, AMD (version 1.2)
- David Yu, Pixar (version 1.0)
- Dean Sekulic, Croteam (version 1.4)
- Dejan Mircevski, Google (version 1.1)
- Dennis Adams, Sony (version 1.4)
- Derek Lamberti, Arm (version 1.4)
- Diego Novillo, Google (versions 1.3, 1.4)
- Dimitris Georgakakis, Think Silicon (version 1.3)
- Dominik Witczak, AMD (versions 1.0, 1.1, 1.3)
- Donald Scorgie, Imagination Technologies (version 1.2)
- Dzmitry Malyshau, Mozilla (versions 1.1, 1.2, 1.3)
- Ed Hutchins, Oculus (version 1.2)
- Eddie Hatfield, Google (version 1.4)
- Eduardo Lima, Igalia (version 1.4)
- Emily Stearns, Khronos (versions 1.2, 1.3, 1.4)

- Enrique de Lucas, Imagination Technologies (version 1.4)
- Eric Sullivan, NVIDIA (version 1.4)
- Faith Ekstrand, Collabora (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Florian Penzkofer, Epic (version 1.4)
- Frank (LingJun) Chen, Qualcomm Technologies, Inc. (version 1.0)
- Frank Yang, Google (version 1.4)
- François Duranleau, Gameloft (versions 1.3, 1.4)
- Fred Liao, Mediatek (version 1.0)
- Gabe Dagani, Samsung Electronics (versions 1.0, 1.4)
- Gabor Sines, AMD (version 1.2)
- Gang Chen, Intel (version 1.4)
- George Erfesoglou, OVR Tech (version 1.4)
- Graeme Leese, Broadcom (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Graham Connor, Imagination Technologies (version 1.0)
- Graham Sellers, Electronic Arts (versions 1.0, 1.1, 1.4)
- Graham Wihlidal, Electronic Arts (version 1.3)
- Greg Fischer, LunarG (versions 1.1, 1.4)
- Greg Roth, Microsoft (version 1.4)
- Gregory Grebe, AMD (versions 1.3, 1.4)
- Hai Nguyen, NVIDIA (versions 1.2, 1.3, 1.4)
- Hamish Todd, Imagination Technologies (version 1.4)
- Hans-Kristian Arntzen, Valve Software (versions 1.1, 1.2, 1.3, 1.4)
- Heman Gala, MAGIX (version 1.4)
- Henri Verbeet, Codeweavers (version 1.2)
- Hugo Devillers, Saarland University (version 1.4)
- Hwanyong Lee, Kyungpook National University (version 1.0)
- Iago Toral, Igalia (versions 1.1, 1.2)
- Ian Ameline, Huawei Technologies Co., Ltd. (version 1.4)
- Ian Elliott, Google (versions 1.0, 1.1, 1.2, 1.4)
- Ian Richardson, Intel (version 1.4)
- Ian Romanick, Intel (versions 1.0, 1.1, 1.3, 1.4)
- Ingrid Kelly, NVIDIA (version 1.4)
- J. D. Rouan, Google (version 1.4)
- Jaebaek Seo, Google (version 1.4)
- Jake Turner, Valve Software (version 1.4)

- Jakob Bornecrantz, Collabora (version 1.4)
- James Fitzpatrick, Imagination Technologies (versions 1.3, 1.4)
- James Hughes, Oculus VR (version 1.0)
- James Jones, NVIDIA (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- James Riordon, Khronos (versions 1.2, 1.3)
- Jamie Madill, Google (versions 1.3, 1.4)
- Jan Hermes, Continental Corporation (versions 1.0, 1.1, 1.4)
- Jan-Harald Fredriksen, Arm (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Jarred Davies, Imagination Technologies (version 1.4)
- Jean Geffroy, id Software (version 1.4)
- Jean-François Roy, Google (versions 1.1, 1.2, 1.3)
- Jean-Noé Morissette, Epic Games (version 1.4)
- Jeff Bolz, NVIDIA (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Jeff Juliano, NVIDIA (versions 1.0, 1.1, 1.2, 1.4)
- Jeff Kiel, NVIDIA (version 1.4)
- Jeff Leger, Qualcomm Technologies, Inc. (versions 1.1, 1.3, 1.4)
- Jeff Phillips, Khronos (versions 1.3, 1.4)
- Jeff Vigil, Samsung Electronics (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Jens Owen, Google (versions 1.0, 1.1)
- Jeremy Gebben, LunarG (version 1.4)
- Jeremy Hayes, LunarG (versions 1.0, 1.4)
- Jesse Barker, Unity Technologies (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Jesse Hall, Google (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Jessica Heerboth, NVIDIA (version 1.4)
- Jingye Wang, Huawei Technologies Co., Ltd. (version 1.4)
- Joe Davis, Samsung Electronics (version 1.1)
- Johannes van Waveren, Oculus VR (versions 1.0, 1.1)
- John Anthony, Arm (versions 1.2, 1.3, 1.4)
- John Kessenich, Google (versions 1.0, 1.1, 1.2, 1.3) (SPIR-V and GLSL for Vulkan spec author)
- John McDonald, Valve Software (versions 1.0, 1.1, 1.2, 1.3)
- John Stone, University of Illinois (version 1.4)
- John Zulauf, LunarG (versions 1.1, 1.2, 1.3)
- Jon Ashburn, LunarG (version 1.0)
- Jon Leech, Independent (versions 1.0, 1.1, 1.2, 1.3, 1.4) (XML toolchain, normative language, release wrangler)

- Jonas Gustavsson, Samsung Electronics (versions 1.0, 1.1)
- Jonas Meyer, Epic Games (versions 1.2, 1.3)
- Jonathan Hamilton, Imagination Technologies (version 1.0)
- Jordan Justen, Intel (version 1.1)
- Jordan Robinson, Samsung Electronics (version 1.4)
- Joshua Ashton, Valve Software (versions 1.3, 1.4)
- Jules Blok, Independent (version 1.4)
- Jungwoo Kim, Samsung Electronics (versions 1.0, 1.1)
- Jörg Wagner, Arm (version 1.1)
- Kalle Raita, Google (version 1.1)
- Karen Ghavam, LunarG (versions 1.1, 1.2, 1.3, 1.4)
- Karl Schultz, LunarG (versions 1.1, 1.2)
- Kathleen Mattson, Khronos (versions 1.0, 1.1, 1.2)
- Kaye Mason, Google (version 1.2)
- Keith Packard, Valve Software (version 1.2)
- Kenneth Benzie, Codeplay Software Ltd. (versions 1.0, 1.1)
- Kenneth Russell, Google (version 1.1)
- Kerch Holt, NVIDIA (versions 1.0, 1.1)
- Kevin O'Neil, AMD (version 1.1)
- Kevin Petit, Arm (versions 1.3, 1.4)
- Kris Rose, Khronos (versions 1.2, 1.3, 1.4)
- Kristian Kristensen, Intel (versions 1.0, 1.1)
- Krzysztof Iwanicki, Samsung Electronics (version 1.0)
- Larry Seiler, Intel (version 1.0)
- Laura Hermanns, Epic Games (version 1.4)
- Laura Shubel, Caster Communications (version 1.3)
- Lauri Ilola, Nokia (version 1.1)
- Lei Zhang, Google (version 1.2)
- Lenny Komow, LunarG (versions 1.1, 1.2)
- Leonid Fedorov, Huawei Technologies Co., Ltd. (version 1.4)
- Liam Middlebrook, NVIDIA (version 1.3)
- Lina Versace, Google (versions 1.0, 1.1, 1.2, 1.4) (system integration subgroup chair)
- Lionel Duc, NVIDIA (version 1.4)
- Lionel Landwerlin, Intel (versions 1.1, 1.2, 1.4)
- Lisie Aartsen, Khronos (versions 1.3, 1.4)

- Liz Maitral, Khronos (version 1.2)
- Lorenzo Dal Col, Khronos (version 1.4) (conformance test subgroup chair)
- Lou Kramer, AMD (version 1.3)
- Lutz Latta, Lucasfilm (version 1.0)
- Lynne Iribarren, Independent (version 1.4)
- Maciej Jesionowski, AMD (version 1.1)
- Mahmoud Kharsa, Samsung Electronics (version 1.4)
- Mais Alnasser, AMD (versions 1.1, 1.4)
- Marcin Kantoich, AMD (version 1.3)
- Marcin Rogucki, Mobica (version 1.1)
- Maria Rovatsou, Codeplay Software Ltd. (version 1.0)
- Mariusz Merecki, Intel (versions 1.3, 1.4)
- Marijn Suijten, Traverse Research (version 1.4)
- Mark Bellamy, Arm (versions 1.2, 1.3, 1.4)
- Mark Callow, Independent (versions 1.0, 1.1, 1.2, 1.3)
- Dhruv Mark Collins, Igalia (version 1.4)
- Mark Kilgard, NVIDIA (versions 1.1, 1.2)
- Mark Lobodzinski, LunarG (versions 1.0, 1.1, 1.2)
- Mark Young, LunarG (versions 1.1, 1.3, 1.4)
- Markus Tavenrath, NVIDIA (version 1.1)
- Marty Johnson, Khronos (versions 1.3, 1.4)
- Mateusz Przybylski, Intel (version 1.0)
- Mathias Heyer, NVIDIA (versions 1.0, 1.1)
- Mathias Schott, NVIDIA (versions 1.0, 1.1)
- Mathieu Robart, Arm (version 1.2)
- Matt Netsch, Qualcomm Technologies, Inc. (versions 1.1, 1.3, 1.4) (conformance test subgroup chair)
- Matthew Rusch, NVIDIA (version 1.3)
- Matthäus Chajdas, Intel (versions 1.1, 1.2, 1.3, 1.4)
- Maurice Ribble, Qualcomm Technologies, Inc. (versions 1.0, 1.1)
- Maxim Lukyanov, Samsung Electronics (version 1.0)
- Maximilien Dagois, Google (version 1.4)
- Michael Antilla, Google (version 1.4)
- Michael Blumenkrantz, Valve Software (versions 1.3, 1.4)
- Michael Lentine, Google (version 1.0)

- Michael O'Hara, AMD (version 1.1)
- Michael Phillip, Samsung Electronics (version 1.2)
- Michael Wong, Codeplay Software Ltd. (version 1.1)
- Michael Worcester, Imagination Technologies (versions 1.0, 1.1)
- Michal Pietrasiuk, Intel (versions 1.0, 1.3, 1.4)
- Mick Beaver, Electronic Arts (version 1.4)
- Mika Isojarvi, Google (versions 1.0, 1.1)
- Mike Schuchardt, LunarG (versions 1.1, 1.2, 1.4)
- Mike Stroyan, LunarG (version 1.0)
- Mike Weiblen, LunarG (versions 1.1, 1.2, 1.3)
- Minyoung Son, Samsung Electronics (version 1.0)
- Mitch Singer, AMD (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Mythri Venugopal, Samsung Electronics (version 1.0)
- Máté Ferenc Nagy-Egri, RasterGrid (version 1.4)
- Natalie Chouinard, Google (version 1.4)
- Nathan Gauër, Google (version 1.4)
- Nathaniel Cesario, LunarG (version 1.4)
- Naveen Leekha, Google (version 1.0)
- Neil Henning, AMD (versions 1.0, 1.1, 1.2, 1.3)
- Neil Hickey, Arm (version 1.2)
- Neil Trevett, NVIDIA (versions 1.0, 1.1, 1.2, 1.3, 1.4) (portability subgroup chair)
- Nick Penwarden, Epic Games (version 1.0)
- Nicola Zaghen, Innosilicon (version 1.4)
- Nicolai Hähnle, AMD (version 1.1)
- Nikhil Joshi, NVIDIA (version 1.4)
- Niklas Smedberg, Unity Technologies (version 1.0)
- Norbert Nopper, Independent (versions 1.0, 1.1)
- Nuno Subtil, NVIDIA (versions 1.1, 1.2, 1.3)
- Pan Gao, Huawei Technologies Co., Ltd. (version 1.4)
- Pat Brown, NVIDIA (version 1.0)
- Patrick Boyle, Qualcomm Technologies, Inc. (version 1.4)
- Patrick Cozzi, Independent (version 1.1)
- Patrick Doane, Activision (versions 1.0, 1.4)
- Patrick Han, Samsung Electronics (version 1.4)
- Patrick Mours, NVIDIA (version 1.4)

- Pau Baiget, Google (version 1.4)
- Peter Kutz, Adobe (version 1.4)
- Peter Lohrmann, AMD (versions 1.0, 1.2)
- Peter Quayle, Imagination Technologies (version 1.4)
- Petros Bantolas, Imagination Technologies (version 1.1)
- Philip Rebohle, Valve Software (version 1.3)
- Pierre Boudier, NVIDIA (versions 1.0, 1.1, 1.2, 1.3)
- Pierre-Loup Griffais, Valve Software (versions 1.0, 1.1, 1.2, 1.3)
- Piers Daniell, NVIDIA (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Ping Liu, Intel (versions 1.3, 1.4)
- Piotr Bialecki, Intel (version 1.0)
- Piotr Byszewski, Mobica (version 1.3)
- Prabindh Sundareson, Samsung Electronics (version 1.0)
- Pradyuman Singh, NVIDIA (version 1.4)
- Pyry Haulos, Google (versions 1.0, 1.1) (conformance test subgroup chair)
- Rachel Bradshaw, Caster Communications (version 1.3)
- Rajeev Rao, Qualcomm Technologies, Inc. (version 1.2)
- Ralph Potter, Samsung Electronics (versions 1.1, 1.2, 1.3, 1.4) (working group chair)
- Ran Wang, Google (version 1.4)
- Raun Krisch, Samsung Electronics (version 1.3)
- Ravi Patlegar, AMD (version 1.4)
- Ravikiran Pallapatula, Huawei Technologies Co., Ltd. (version 1.4)
- Ray Smith, Arm (versions 1.0, 1.1, 1.2)
- Ricardo Garcia, Igalia (versions 1.3, 1.4)
- Richard Huddy, Samsung Electronics (versions 1.2, 1.3)
- Richard Van Natta, Firaxis (version 1.4)
- Richard Wang, Huawei Technologies Co., Ltd. (version 1.4)
- Rick Hammerstone, Qualcomm Technologies, Inc. (version 1.4)
- Rob Barris, NVIDIA (version 1.1)
- Rob Stepinski, Transgaming (version 1.0)
- Robert Simpson, Qualcomm Technologies, Inc. (versions 1.0, 1.1, 1.3, 1.4)
- Rohan Garg, Intel (version 1.4)
- Rolando Caloca Olivares, Epic Games (versions 1.0, 1.1, 1.2, 1.3)
- Ronan Keryell, Xilinx (versions 1.3, 1.4)
- Roy Ju, Mediatek (version 1.0)

- Rufus Hamade, Imagination Technologies (version 1.0)
- Ruihao Zhang, Qualcomm Technologies, Inc. (versions 1.1, 1.2, 1.3, 1.4)
- SU Fujia, Huawei Technologies Co., Ltd. (version 1.4)
- Sam Martin, Arm (version 1.4)
- Samuel (Sheng-Wen) Huang, Mediatek (versions 1.3, 1.4)
- Samuel Iglesias Gonsalvez, Igalia (version 1.3)
- Samuel Pitoiset, Valve Software (version 1.4)
- Sandeep Kakarlapudi, Arm (version 1.4)
- Sascha Willems, Self (versions 1.3, 1.4)
- Sean Ellis, Arm (version 1.0)
- Sean Harmer, KDAB Group (versions 1.0, 1.1)
- Serdar Kocdemir, Google (version 1.4)
- Sergey Solovyev, Tencent (version 1.4)
- Seth Williams, NVIDIA (version 1.4)
- Shahbaz Youssefi, Google (version 1.4)
- Shannon Woods, NVIDIA (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Shaun Heald, Innosilicon (version 1.4)
- Sinuhe Hardegree, Samsung Electronics (version 1.4)
- Slawomir Cygan, Intel (versions 1.0, 1.1, 1.3, 1.4)
- Slawomir Grajewski, Intel (versions 1.0, 1.1, 1.3, 1.4)
- Sorel Bosan, AMD (version 1.1)
- Soroush Kashani, Imagination Technologies (version 1.4)
- Spencer Fricke, LunarG (versions 1.2, 1.3, 1.4)
- Stefanus Du Toit, Google (version 1.0)
- Stephane Cerveau, Igalia (version 1.4)
- Stephen Huang, Mediatek (version 1.1)
- Steve Hill, Broadcom (versions 1.0, 1.2)
- Steve Viggers, Core Avionics & Industrial Inc. (versions 1.0, 1.2)
- Steve Winston, Holochip (versions 1.3, 1.4)
- Steven Gu, Huawei Technologies Co., Ltd. (version 1.4)
- Steven Perron, Google (version 1.4)
- Stuart Smith, AMD (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Sujeevan Rajayogam, Google (version 1.3)
- Theresa Foley, Intel (version 1.0)
- Thomas Caissard, Adobe (version 1.4)

- Tiago Costa, Epic Games (version 1.4)
- Tilmann Scheller, Samsung Electronics (version 1.1)
- Tim Lewis, Khronos (versions 1.3, 1.4)
- Timo Suoranta, AMD (version 1.0)
- Timothy Lottes, AMD (versions 1.0, 1.1)
- Timur Kristof, Valve Software (version 1.4)
- Ting Wei, Arm (version 1.4)
- Tobias Hector, AMD (versions 1.0, 1.1, 1.2, 1.3, 1.4) (validity language and toolchain)
- Tobin Ehli, LunarG (version 1.0)
- Tom Olson, Arm (versions 1.0, 1.1, 1.2, 1.3, 1.4) (working group chair)
- Tomasz Bednarz, Independent (version 1.1)
- Tomasz Kubale, Intel (version 1.0)
- Tony Barbour, LunarG (versions 1.0, 1.1, 1.2)
- Tony Zlatinski, NVIDIA (versions 1.3, 1.4)
- Trevor Black, Google (version 1.4)
- Tyler Nowicki, AMD (version 1.4)
- Vassili Nikolaev, NVIDIA (version 1.4)
- Vatsalya Prasad, Huawei Technologies Co., Ltd. (version 1.4)
- Victor Eruhimov, Unknown (version 1.1)
- Vikram Kushwaha, NVIDIA (versions 1.3, 1.4)
- Vikram Tarikere, Imagination Technologies (version 1.4)
- Vincent Hindriksen, Stream HPC (versions 1.2, 1.3)
- Víctor Jáquez, Igalia (version 1.4)
- Wang Jingye, Huawei Technologies Co., Ltd. (version 1.4)
- Wasim Abbas, Arm (version 1.3)
- Wayne Lister, Imagination Technologies (version 1.0)
- William Wheeler, Imagination Technologies (version 1.4)
- Wolfgang Engel, Unknown (version 1.1)
- Wooyoung Kim, Qualcomm Technologies, Inc. (version 1.4)
- Wyvern Wang, Huawei Technologies Co., Ltd. (version 1.3)
- Yajun Zhang, Tencent (version 1.4)
- Yanjun Zhang, VeriSilicon (versions 1.0, 1.1, 1.2, 1.3, 1.4)
- Youngwang Kim, Arm (version 1.4)
- Yu Chang Wang, Huawei Technologies Co., Ltd. (version 1.4)
- Yuehai Du, Qualcomm Technologies, Inc. (version 1.4)

- Yunxing Zhu, Huawei Technologies Co., Ltd. (version 1.3)
- Yury Uralsky, NVIDIA (version 1.4)
- Zehui Lin, Huawei Technologies Co., Ltd. (version 1.4)
- Žiga Markuš, LunarG (version 1.4)

Other Credits

The Vulkan Advisory Panel members provided important real-world usage information and advice that helped guide design decisions.

The wider Vulkan community have provided useful feedback, questions and specification changes that have helped improve the quality of the Specification via [GitHub](#).

Administrative support to the Working Group for Vulkan 1.1 and beyond was provided by Khronos staff including Ann Thorsnes, Blaine Kohl, Dennis Fu, Dominic Agoro-Ombaka, Emily Stearns, Jeff Phillips, Kendra Casper, Lisie Aartsen, Liz Maitral, Marty Johnson, Tim Lewis, and Xiao-Yu CHENG; and by Alex Crabb, Laura Shubel, and Rachel Bradshaw of Caster Communications.

Administrative support for Vulkan 1.0 was provided by Andrew Riegel, Elizabeth Riegel, Glenn Fredericks, Kathleen Mattson and Michelle Clark of Gold Standard Group.

Technical support was provided by James Riordon, site administration of Khronos.org and OpenGL.org.