Vulkan API Reference Pages

Version 1.0.55, Mon, 17 Jul 2017 00:48:15 +0000

Table of Contents

Copyright

Vulkan Commands

vkAllocateCommandBuffers(3)

Name

vkAllocateCommandBuffers - Allocate command buffers from an existing command pool

C Specification

To allocate command buffers, call:

```
VkResult vkAllocateCommandBuffers(
    VkDevice                                    device,
    const VkCommandBufferAllocateInfo*          pAllocateInfo,
    VkCommandBuffer*                            pCommandBuffers);
```

Parameters

- `device` is the logical device that owns the command pool.
- `pAllocateInfo` is a pointer to an instance of the `VkCommandBufferAllocateInfo` structure describing parameters of the allocation.
- `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.

Description

When command buffers are first allocated, they are in the initial state.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pAllocateInfo` **must** be a pointer to a valid `VkCommandBufferAllocateInfo` structure
- `pCommandBuffers` **must** be a pointer to an array of `pAllocateInfo`::commandBufferCount `VkCommandBuffer` handles

Host Synchronization

- Host access to `pAllocateInfo`::commandPool **must** be externally synchronized

Return Codes

Success

- VK_SUCCESS

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

See Also

VkCommandBuffer, VkCommandBufferAllocateInfo, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkAllocateCommandBuffers

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkAllocateDescriptorSets(3)

Name

vkAllocateDescriptorSets - Allocate one or more descriptor sets

C Specification

To allocate descriptor sets from a descriptor pool, call:

```
VkResult vkAllocateDescriptorSets(
    VkDevice                                    device,
    const VkDescriptorSetAllocateInfo*          pAllocateInfo,
    VkDescriptorSet*                            pDescriptorSets);
```

Parameters

- `device` is the logical device that owns the descriptor pool.
- `pAllocateInfo` is a pointer to an instance of the 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 array **must** be at least the length specified by the `descriptorSetCount` member of `pAllocateInfo`.

Description

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. However, the descriptor set **can** be bound in a command buffer without causing errors or exceptions. All entries that are statically used by a pipeline in a drawing or dispatching command **must** have been populated before the descriptor set is bound for use by that command. Entries that are not statically used by a pipeline **can** have uninitialized descriptors or descriptors of resources that have been destroyed, and executing a draw or dispatch with such a descriptor set bound does not cause undefined behavior. This means applications need not populate unused entries with dummy descriptors.

If an allocation fails due to fragmentation, an indeterminate error is returned with an unspecified error code. Any returned error other than `VK_ERROR_FRAGMENTED_POOL` does not imply its usual meaning: applications **should** assume that the allocation failed due to fragmentation, and create a new descriptor pool.

Note

Applications **should** check for a negative return value when allocating new descriptor sets, assume that any error effectively means `VK_ERROR_FRAGMENTED_POOL`, and try to create a new descriptor pool. If `VK_ERROR_FRAGMENTED_POOL` is the actual return value, it adds certainty to that decision.

The reason for this is that `VK_ERROR_FRAGMENTED_POOL` was only added in a later revision of the 1.0 specification, and so drivers **may** return other errors if they were written against earlier revisions. To ensure full compatibility with earlier patch revisions, these other errors are allowed.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pAllocateInfo` **must** be a pointer to a valid `VkDescriptorSetAllocateInfo` structure
- `pDescriptorSets` **must** be a pointer to an array of `pAllocateInfo`::descriptorSetCount `VkDescriptorSet` handles

Host Synchronization

- Host access to `pAllocateInfo`::descriptorPool **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_FRAGMENTED_POOL`

See Also

VkDescriptorSet, VkDescriptorSetAllocateInfo, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkAllocateDescriptorSets

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkAllocateMemory(3)

Name

vkAllocateMemory - Allocate GPU memory

C Specification

To allocate memory objects, call:

```
VkResult vkAllocateMemory(
    VkDevice                                    device,
    const VkMemoryAllocateInfo*                 pAllocateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDeviceMemory*                             pMemory);
```

Parameters

- `device` is the logical device that owns the memory.
- `pAllocateInfo` is a pointer to an instance of the VkMemoryAllocateInfo structure describing parameters of the allocation. A successful 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.

Description

Allocations returned by `vkAllocateMemory` are guaranteed to meet any alignment requirement by 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.

There is an implementation-dependent maximum number of memory allocations which **can** be simultaneously created on a device. This is specified by the `maxMemoryAllocationCount` member of the `VkPhysicalDeviceLimits` structure. If `maxMemoryAllocationCount` is exceeded, `vkAllocateMemory` will return `VK_ERROR_TOO_MANY_OBJECTS`.

> *Note*
>
> 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` **should** be returned.

Valid Usage

- The number of currently valid memory objects, allocated from `device`, **must** be less than `VkPhysicalDeviceLimits`::`maxMemoryAllocationCount`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pAllocateInfo` **must** be a pointer to a valid `VkMemoryAllocateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pMemory` **must** be a pointer to a `VkDeviceMemory` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_TOO_MANY_OBJECTS`

See Also

VkAllocationCallbacks, VkDevice, VkDeviceMemory, VkMemoryAllocateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkAllocateMemory

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkBeginCommandBuffer(3)

Name

vkBeginCommandBuffer - Start recording a command buffer

C Specification

To begin recording a command buffer, call:

```
VkResult vkBeginCommandBuffer(
    VkCommandBuffer                             commandBuffer,
    const VkCommandBufferBeginInfo*             pBeginInfo);
```

Parameters

- `commandBuffer` is the handle of the command buffer which is to be put in the recording state.
- `pBeginInfo` is an instance of the `VkCommandBufferBeginInfo` structure, which defines additional information about how the command buffer begins recording.

Description

Valid Usage

- `commandBuffer` **must** not be in the recording or pending state.
- 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.
- If `commandBuffer` is a secondary command buffer, the `pInheritanceInfo` member of `pBeginInfo` **must** be a valid `VkCommandBufferInheritanceInfo` structure
- If `commandBuffer` is a secondary command buffer and either the `occlusionQueryEnable` member of the `pInheritanceInfo` member of `pBeginInfo` is `VK_FALSE`, or the precise occlusion queries feature is not enabled, the `queryFlags` member of the `pInheritanceInfo` member `pBeginInfo` **must** not contain `VK_QUERY_CONTROL_PRECISE_BIT`

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `pBeginInfo` **must** be a 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_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkCommandBuffer, VkCommandBufferBeginInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkBeginCommandBuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkBindBufferMemory(3)

Name

vkBindBufferMemory - Bind device memory to a buffer object

C Specification

To attach memory to a buffer object, call:

```
VkResult vkBindBufferMemory(
    VkDevice                                    device,
    VkBuffer                                    buffer,
    VkDeviceMemory                              memory,
    VkDeviceSize                                memoryOffset);
```

Parameters

- `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.

Description

Valid Usage

- `buffer` **must** not already be backed by a memory object
- `buffer` **must** not have been created with any sparse memory binding flags
- `memoryOffset` **must** be less than the size of `memory`
- If `buffer` was created with the `VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT` or `VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT`, `memoryOffset` **must** be a multiple of `VkPhysicalDeviceLimits`::`minTexelBufferOffsetAlignment`
- If `buffer` was created with the `VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT`, `memoryOffset` **must** be a multiple of `VkPhysicalDeviceLimits`::`minUniformBufferOffsetAlignment`
- If `buffer` was created with the `VK_BUFFER_USAGE_STORAGE_BUFFER_BIT`, `memoryOffset` **must** be a multiple of `VkPhysicalDeviceLimits`::`minStorageBufferOffsetAlignment`
- `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`
- `memoryOffset` **must** be an integer multiple of the `alignment` member of the `VkMemoryRequirements` structure returned from a call to `vkGetBufferMemoryRequirements` with `buffer`
- The `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`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `buffer` **must** be a valid `VkBuffer` handle
- `memory` **must** be a valid `VkDeviceMemory` handle
- `buffer` **must** have been created, allocated, or retrieved from `device`
- `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_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkBuffer, VkDevice, VkDeviceMemory, `VkDeviceSize`

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkBindBufferMemory

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkBindImageMemory(3)

Name

vkBindImageMemory - Bind device memory to an image object

C Specification

To attach memory to an image object, call:

```
VkResult vkBindImageMemory(
    VkDevice                                    device,
    VkImage                                     image,
    VkDeviceMemory                              memory,
    VkDeviceSize                                memoryOffset);
```

Parameters

- `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.

Description

Valid Usage

- `image` **must** not already be backed by a memory object
- `image` **must** not have been created with any sparse memory binding flags
- `memoryOffset` **must** be less than the size of `memory`
- `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`
- `memoryOffset` **must** be an integer multiple of the `alignment` member of the `VkMemoryRequirements` structure returned from a call to `vkGetImageMemoryRequirements` with `image`
- The `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)

- `device` **must** be a valid `VkDevice` handle
- `image` **must** be a valid `VkImage` handle
- `memory` **must** be a valid `VkDeviceMemory` handle
- `image` **must** have been created, allocated, or retrieved from `device`
- `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_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkDevice, VkDeviceMemory, `VkDeviceSize`, VkImage

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkBindImageMemory

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdBeginQuery(3)

Name

vkCmdBeginQuery - Begin a query

C Specification

To begin a query, call:

```
void vkCmdBeginQuery(
    VkCommandBuffer                             commandBuffer,
    VkQueryPool                                 queryPool,
    uint32_t                                    query,
    VkQueryControlFlags                         flags);
```

Parameters

- `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.

Description

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.

Valid Usage

- The query identified by `queryPool` and `query` **must** currently not be active
- The query identified by `queryPool` and `query` **must** be unavailable
- If the precise occlusion queries 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`
- `queryPool` **must** have been created with a `queryType` that differs from that of any other queries that have been made active, and are currently still active within `commandBuffer`
- `query` **must** be less than the number of queries in `queryPool`
- If the `queryType` used to create `queryPool` was `VK_QUERY_TYPE_OCCLUSION`, the `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- 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
- 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

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `queryPool` **must** be a valid `VkQueryPool` handle
- `flags` **must** be a valid combination of VkQueryControlFlagBits values
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- 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	Pipeline Type
Primary Secondary	Both	Graphics compute	

See Also

VkCommandBuffer, VkQueryControlFlags, VkQueryPool

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdBeginQuery

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdBeginRenderPass(3)

Name

vkCmdBeginRenderPass - Begin a new render pass

C Specification

To begin a render pass instance, call:

```
void vkCmdBeginRenderPass(
    VkCommandBuffer                             commandBuffer,
    const VkRenderPassBeginInfo*                pRenderPassBegin,
    VkSubpassContents                           contents);
```

Parameters

- `commandBuffer` is the command buffer in which to record the command.
- `pRenderPassBegin` is a pointer to a VkRenderPassBeginInfo structure (defined below) which indicates 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.

Description

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

- 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 subresource of the framebuffer specified in the framebuffer member of pRenderPassBegin **must** have been created with VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT set
- 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_STENCIL_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL then the corresponding attachment image subresource of the framebuffer specified in the framebuffer member of pRenderPassBegin **must** have been created with VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT set
- 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 subresource of the framebuffer specified in the framebuffer member of pRenderPassBegin **must** have been created with VK_IMAGE_USAGE_SAMPLED_BIT or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT set
- 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 subresource of the framebuffer specified in the framebuffer member of pRenderPassBegin **must** have been created with VK_IMAGE_USAGE_TRANSFER_SRC_BIT set
- 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 subresource of the framebuffer specified in the framebuffer member of pRenderPassBegin **must** have been created with VK_IMAGE_USAGE_TRANSFER_DST_BIT set
- If any of the initialLayout members 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
- The srcStageMask and 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.

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `pRenderPassBegin` **must** be a pointer to a valid `VkRenderPassBeginInfo` structure
- `contents` **must** be a valid VkSubpassContents value
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- This command **must** only be called outside of a render pass instance
- `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	Pipeline Type
Primary	Outside	Graphics	Graphics

See Also

VkCommandBuffer, VkRenderPassBeginInfo, VkSubpassContents

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdBeginRenderPass

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdBindDescriptorSets(3)

Name

vkCmdBindDescriptorSets - Binds descriptor sets to a command buffer

C Specification

To bind one or more descriptor sets to a command buffer, call:

```
void vkCmdBindDescriptorSets(
    VkCommandBuffer                             commandBuffer,
    VkPipelineBindPoint                         pipelineBindPoint,
    VkPipelineLayout                            layout,
    uint32_t                                    firstSet,
    uint32_t                                    descriptorSetCount,
    const VkDescriptorSet*                      pDescriptorSets,
    uint32_t                                    dynamicOffsetCount,
    const uint32_t*                             pDynamicOffsets);
```

Parameters

- `commandBuffer` is the command buffer that the descriptor sets will be bound to.
- `pipelineBindPoint` is a VkPipelineBindPoint indicating whether the descriptors will be used by graphics pipelines or compute pipelines. There is a separate set of bind points for each of graphics and compute, so binding one does not disturb the other.
- `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 an array of handles to `VkDescriptorSet` objects describing the descriptor sets to write 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.

Description

`vkCmdBindDescriptorSets` causes the sets numbered [`firstSet`.. `firstSet`+`descriptorSetCount`-1] to use the bindings stored in `pDescriptorSets`[0..`descriptorSetCount`-1] for subsequent rendering commands (either compute or graphics, according to the `pipelineBindPoint`). Any bindings that were previously applied via these sets are no longer valid.

Once bound, a descriptor set affects rendering of subsequent graphics or compute commands in the command buffer until a different set is bound to the same set number, or else until 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 draw or dispatch 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.

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 length 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 graphics or compute commands, as defined in the Pipeline Layout Compatibility section.

The descriptor set contents bound by a call to `vkCmdBindDescriptorSets` **may** be consumed during host execution of the command, or during shader execution of the resulting draws, or any time in between. Thus, the contents **must** not be altered (overwritten by an update command, or freed) between when the command is recorded 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

- Any given 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`
- `dynamicOffsetCount` **must** be equal to the total number of dynamic descriptors in `pDescriptorSets`
- The sum of `firstSet` and `descriptorSetCount` **must** be less than or equal to `VkPipelineLayoutCreateInfo`::`setLayoutCount` provided when `layout` was created
- `pipelineBindPoint` **must** be supported by the `commandBuffer`'s parent `VkCommandPool`'s queue family
- Any given element of `pDynamicOffsets` **must** satisfy the required alignment for the corresponding descriptor binding's descriptor type

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `pipelineBindPoint` **must** be a valid VkPipelineBindPoint value
- `layout` **must** be a valid `VkPipelineLayout` handle
- `pDescriptorSets` **must** be a pointer to an array of `descriptorSetCount` valid `VkDescriptorSet` handles
- If `dynamicOffsetCount` is not `0`, `pDynamicOffsets` **must** be a pointer to an array of `dynamicOffsetCount` `uint32_t` values
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- `descriptorSetCount` **must** be greater than `0`
- Each of `commandBuffer`, `layout`, and the elements of `pDescriptorSets` **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	Pipeline Type
Primary Secondary	Both	Graphics compute	

See Also

VkCommandBuffer, VkDescriptorSet, VkPipelineBindPoint, VkPipelineLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdBindDescriptorSets

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdBindIndexBuffer(3)

Name

vkCmdBindIndexBuffer - Bind an index buffer to a command buffer

C Specification

To bind an index buffer to a command buffer, call:

```
void vkCmdBindIndexBuffer(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    VkIndexType                                 indexType);
```

Parameters

- `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 whether indices are treated as 16 bits or 32 bits.

Description

Valid Usage

- `offset` **must** be less than the size of `buffer`
- The sum of `offset` and the address of the range of `VkDeviceMemory` object that is backing `buffer`, **must** be a multiple of the type indicated by `indexType`
- `buffer` **must** have been created with the `VK_BUFFER_USAGE_INDEX_BUFFER_BIT` flag
- If `buffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `buffer` **must** be a valid `VkBuffer` handle
- `indexType` **must** be a valid VkIndexType value
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- 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	Pipeline Type
Primary Secondary	Both	Graphics	

See Also

VkBuffer, VkCommandBuffer, `VkDeviceSize`, VkIndexType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdBindIndexBuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdBindPipeline(3)

Name

vkCmdBindPipeline - Bind a pipeline object to a command buffer

C Specification

Once a pipeline has been created, it **can** be bound to the command buffer using the command:

```
void vkCmdBindPipeline(
    VkCommandBuffer                             commandBuffer,
    VkPipelineBindPoint                         pipelineBindPoint,
    VkPipeline                                  pipeline);
```

Parameters

- `commandBuffer` is the command buffer that the pipeline will be bound to.
- `pipelineBindPoint` is a VkPipelineBindPoint value specifying whether to bind to the compute or graphics bind point. Binding one does not disturb the other.
- `pipeline` is the pipeline to be bound.

Description

Once bound, a pipeline binding affects subsequent graphics or compute commands in the command buffer until a different pipeline is bound to the bind point. The pipeline bound to `VK_PIPELINE_BIND_POINT_COMPUTE` controls the behavior of vkCmdDispatch and vkCmdDispatchIndirect. The pipeline bound to `VK_PIPELINE_BIND_POINT_GRAPHICS` controls the behavior of vkCmdDraw, vkCmdDrawIndexed, vkCmdDrawIndirect, and vkCmdDrawIndexedIndirect. No other commands are affected by the pipeline state.

Valid Usage

- If `pipelineBindPoint` is `VK_PIPELINE_BIND_POINT_COMPUTE`, the `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- If `pipelineBindPoint` is `VK_PIPELINE_BIND_POINT_GRAPHICS`, the `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- If `pipelineBindPoint` is `VK_PIPELINE_BIND_POINT_COMPUTE`, `pipeline` **must** be a compute pipeline
- If `pipelineBindPoint` is `VK_PIPELINE_BIND_POINT_GRAPHICS`, `pipeline` **must** be a graphics pipeline
- If the variable multisample rate feature is not supported, `pipeline` is a graphics pipeline, the current subpass has 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

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `pipelineBindPoint` **must** be a valid VkPipelineBindPoint value
- `pipeline` **must** be a valid `VkPipeline` handle
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- 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	Pipeline Type
Primary Secondary	Both	Graphics compute	

See Also

VkCommandBuffer, VkPipeline, VkPipelineBindPoint

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdBindPipeline

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdBindVertexBuffers(3)

Name

vkCmdBindVertexBuffers - Bind vertex buffers to a command buffer

C Specification

To bind vertex buffers to a command buffer for use in subsequent draw commands, call:

```
void vkCmdBindVertexBuffers(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstBinding,
    uint32_t                                    bindingCount,
    const VkBuffer*                             pBuffers,
    const VkDeviceSize*                         pOffsets);
```

Parameters

- `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.

Description

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 draw commands.

Valid Usage

- `firstBinding` **must** be less than `VkPhysicalDeviceLimits`::`maxVertexInputBindings`
- The sum of `firstBinding` and `bindingCount` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxVertexInputBindings`
- All elements of `pOffsets` **must** be less than the size of the corresponding element in `pBuffers`
- All elements of `pBuffers` **must** have been created with the `VK_BUFFER_USAGE_VERTEX_BUFFER_BIT` flag
- Each element of `pBuffers` that is non-sparse **must** be bound completely and contiguously to a single `VkDeviceMemory` object

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `pBuffers` **must** be a pointer to an array of `bindingCount` valid `VkBuffer` handles
- `pOffsets` **must** be a pointer to an array of `bindingCount` `VkDeviceSize` values
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- `bindingCount` **must** be greater than `0`
- Both of `commandBuffer`, and the elements of `pBuffers` **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	Pipeline Type
Primary Secondary	Both	Graphics	

See Also

VkBuffer, VkCommandBuffer, `VkDeviceSize`

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdBindVertexBuffers

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdBlitImage(3)

Name

vkCmdBlitImage - Copy regions of an image, potentially performing format conversion,

C Specification

To copy regions of a source image into a destination image, potentially performing format conversion, arbitrary scaling, and filtering, call:

```
void vkCmdBlitImage(
    VkCommandBuffer                             commandBuffer,
    VkImage                                     srcImage,
    VkImageLayout                               srcImageLayout,
    VkImage                                     dstImage,
    VkImageLayout                               dstImageLayout,
    uint32_t                                    regionCount,
    const VkImageBlit*                          pRegions,
    VkFilter                                    filter);
```

Parameters

- `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.

Description

`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:

  $u_{base} = i + ½$

$v_{base} = j + ½$

$w_{base} = k + ½$

- These base coordinates are then offset by the first destination offset:

$u_{offset} = u_{base} - x_{dst0}$

$v_{offset} = v_{base} - y_{dst0}$

$w_{offset} = w_{base} - z_{dst0}$

$a_{offset} = a - \texttt{baseArrayCount}_{dst}$

- The scale is determined from the source and destination regions, and applied to the offset coordinates:

scale_u = $(x_{src1} - x_{src0}) / (x_{dst1} - x_{dst0})$

scale_v = $(y_{src1} - y_{src0}) / (y_{dst1} - y_{dst0})$

scale_w = $(z_{src1} - z_{src0}) / (z_{dst1} - z_{dst0})$

$u_{scaled} = u_{offset} * scale_u$

$v_{scaled} = v_{offset} * scale_v$

$w_{scaled} = w_{offset} * scale_w$

- Finally the source offset is added to the scaled coordinates, to determine the final unnormalized coordinates used to sample from `srcImage`:

$u = u_{scaled} + x_{src0}$

$v = v_{scaled} + y_{src0}$

$w = w_{scaled} + z_{src0}$

q = `mipLevel`

$a = a_{offset} + \texttt{baseArrayCount}_{src}$

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.

3D textures are blitted slice by slice. Slices in the source region bounded by `srcOffsets`[0].`z` and `srcOffsets`[1].`z` are copied to slices in the destination region bounded by `dstOffsets`[0].`z` and `dstOffsets`[1].`z`. For each destination slice, a source z coordinate is linearly interpolated between `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. If `filter` parameter is `VK_FILTER_NEAREST` then value sampled from the source image is taken from the single nearest slice (with undefined 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, unscaled 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

- The source region specified by a given element of pRegions **must** be a region that is contained within srcImage
- The destination region specified by a given element of pRegions **must** be a region that is contained within dstImage
- 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
- srcImage **must** use a format that supports VK_FORMAT_FEATURE_BLIT_SRC_BIT, which is indicated by VkFormatProperties::linearTilingFeatures (for linearly tiled images) or VkFormatProperties::optimalTilingFeatures (for optimally tiled images) - as returned by vkGetPhysicalDeviceFormatProperties
- srcImage **must** have been created with VK_IMAGE_USAGE_TRANSFER_SRC_BIT usage flag
- If srcImage is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- srcImageLayout **must** specify the layout of the image subresources of srcImage specified in pRegions at the time this command is executed on a VkDevice
- srcImageLayout **must** be VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL or VK_IMAGE_LAYOUT_GENERAL
- dstImage **must** use a format that supports VK_FORMAT_FEATURE_BLIT_DST_BIT, which is indicated by VkFormatProperties::linearTilingFeatures (for linearly tiled images) or VkFormatProperties::optimalTilingFeatures (for optimally tiled images) - as returned by vkGetPhysicalDeviceFormatProperties
- dstImage **must** have been created with VK_IMAGE_USAGE_TRANSFER_DST_BIT usage flag
- If dstImage is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- dstImageLayout **must** specify the layout of the image subresources of dstImage specified in pRegions at the time this command is executed on a VkDevice
- dstImageLayout **must** be VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL or VK_IMAGE_LAYOUT_GENERAL
- The sample count of srcImage and dstImage **must** both be equal to VK_SAMPLE_COUNT_1_BIT
- 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
- 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
- If either of srcImage or dstImage was created with a depth/stencil format, the other **must** have exactly the same format
- If srcImage was created with a depth/stencil format, filter **must** be VK_FILTER_NEAREST
- srcImage **must** have been created with a samples value of VK_SAMPLE_COUNT_1_BIT
- dstImage **must** have been created with a samples value of VK_SAMPLE_COUNT_1_BIT
- If filter is VK_FILTER_LINEAR, srcImage **must** be of a format which supports linear filtering, as specified by the VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT flag in

VkFormatProperties::linearTilingFeatures (for a linear image) or VkFormatProperties::optimalTilingFeatures(for an optimally tiled image) returned by vkGetPhysicalDeviceFormatProperties

Valid Usage (Implicit)

- commandBuffer **must** be a valid VkCommandBuffer handle
- srcImage **must** be a valid VkImage handle
- srcImageLayout **must** be a valid VkImageLayout value
- dstImage **must** be a valid VkImage handle
- dstImageLayout **must** be a valid VkImageLayout value
- pRegions **must** be a pointer to an array of regionCount valid VkImageBlit structures
- filter **must** be a valid VkFilter value
- commandBuffer **must** be in the recording state
- The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- This command **must** only be called outside of a render pass instance
- regionCount **must** be greater than 0
- 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	Pipeline Type
Primary Secondary	Outside	Graphics	Transfer

See Also

VkCommandBuffer, VkFilter, VkImage, VkImageBlit, VkImageLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdBlitImage

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdClearAttachments(3)

Name

vkCmdClearAttachments - Clear regions within currently bound framebuffer attachments

C Specification

To clear one or more regions of color and depth/stencil attachments inside a render pass instance, call:

```
void vkCmdClearAttachments(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    attachmentCount,
    const VkClearAttachment*                    pAttachments,
    uint32_t                                    rectCount,
    const VkClearRect*                          pRects);
```

Parameters

- `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` points to an array of VkClearRect structures defining regions within each selected attachment to clear.

Description

`vkCmdClearAttachments` **can** clear multiple regions of each attachment used in the current subpass of a render pass instance. This command **must** be called only inside a render pass instance, and implicitly selects the images to clear based on the current framebuffer attachments and the command parameters.

Valid Usage

- If the `aspectMask` member of any given element of `pAttachments` contains `VK_IMAGE_ASPECT_COLOR_BIT`, the `colorAttachment` member of those elements **must** refer to a valid color attachment in the current subpass
- The rectangular region specified by a given element of `pRects` **must** be contained within the render area of the current render pass instance
- The layers specified by a given element of `pRects` **must** be contained within every attachment that `pAttachments` refers to

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `pAttachments` **must** be a pointer to an array of `attachmentCount` valid `VkClearAttachment` structures
- `pRects` **must** be a pointer to an array of `rectCount` `VkClearRect` structures
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- This command **must** only be called inside of a render pass instance
- `attachmentCount` **must** be greater than `0`
- `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	Pipeline Type
Primary Secondary	Inside	Graphics	Graphics

See Also

VkClearAttachment, VkClearRect, VkCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdClearAttachments

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdClearColorImage(3)

Name

vkCmdClearColorImage - Clear regions of a color image

C Specification

To clear one or more subranges of a color image, call:

```
void vkCmdClearColorImage(
    VkCommandBuffer                             commandBuffer,
    VkImage                                     image,
    VkImageLayout                               imageLayout,
    const VkClearColorValue*                    pColor,
    uint32_t                                    rangeCount,
    const VkImageSubresourceRange*              pRanges);
```

Parameters

- `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`.
- `pColor` is a pointer to a VkClearColorValue structure that contains the values the image subresource ranges will be cleared to (see html/vkspec.html#clears-values below).
- `rangeCount` is the number of image subresource range structures in `pRanges`.
- `pRanges` points to an array of VkImageSubresourceRange structures that describe a range of mipmap levels, array layers, and aspects to be cleared, as described in Image Views. The `aspectMask` of all image subresource ranges **must** only include `VK_IMAGE_ASPECT_COLOR_BIT`.

Description

Each specified range in `pRanges` is cleared to the value specified by `pColor`.

Valid Usage

- `image` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag
- If `image` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- `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`
- `imageLayout` **must** be `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- 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
- If the VkImageSubresourceRange::`levelCount` member of any element of the `pRanges` array is not `VK_REMAINING_MIP_LEVELS`, it **must** be non-zero and VkImageSubresourceRange ::`baseMipLevel` + VkImageSubresourceRange::`levelCount` for that element of the `pRanges` array **must** be less than or equal to the `mipLevels` specified in VkImageCreateInfo when `image` was created
- 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
- If the VkImageSubresourceRange::`layerCount` member of any element of the `pRanges` array is not `VK_REMAINING_ARRAY_LAYERS`, it **must** be non-zero and VkImageSubresourceRange ::`baseArrayLayer` + VkImageSubresourceRange::`layerCount` for that element of the `pRanges` array **must** be less than or equal to the `arrayLayers` specified in VkImageCreateInfo when `image` was created
- `image` **must** not have a compressed or depth/stencil format

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `image` **must** be a valid `VkImage` handle
- `imageLayout` **must** be a valid VkImageLayout value
- `pColor` **must** be a pointer to a valid `VkClearColorValue` union
- `pRanges` **must** be a pointer to an array of `rangeCount` valid `VkImageSubresourceRange` structures
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- This command **must** only be called outside of a render pass instance
- `rangeCount` **must** be greater than `0`
- 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	Pipeline Type
Primary Secondary	Outside	Graphics compute	Transfer

See Also

VkClearColorValue, VkCommandBuffer, VkImage, VkImageLayout, VkImageSubresourceRange

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdClearColorImage

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdClearDepthStencilImage(3)

Name

vkCmdClearDepthStencilImage - Fill regions of a combined depth/stencil image

C Specification

To clear one or more subranges of a depth/stencil image, call:

```
void vkCmdClearDepthStencilImage(
    VkCommandBuffer                             commandBuffer,
    VkImage                                     image,
    VkImageLayout                               imageLayout,
    const VkClearDepthStencilValue*             pDepthStencil,
    uint32_t                                    rangeCount,
    const VkImageSubresourceRange*              pRanges);
```

Parameters

- `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 that contains the values the depth and stencil image subresource ranges will be cleared to (see html/vkspec.html#clears-values below).
- `rangeCount` is the number of image subresource range structures in `pRanges`.
- `pRanges` points to an array of VkImageSubresourceRange structures that describe a range of mipmap levels, array layers, and aspects to be cleared, as described in Image Views. The `aspectMask` of each image subresource range in `pRanges` **can** include `VK_IMAGE_ASPECT_DEPTH_BIT` if the image format has a depth component, and `VK_IMAGE_ASPECT_STENCIL_BIT` if the image format has a stencil component. `pDepthStencil` is a pointer to a `VkClearDepthStencilValue` structure that contains the values the image subresource ranges will be cleared to (see html/vkspec.html#clears-values below).

Description

Valid Usage

- `image` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag
- If `image` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- `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`
- `imageLayout` **must** be either of `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- 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
- If the VkImageSubresourceRange::`levelCount` member of any element of the `pRanges` array is not `VK_REMAINING_MIP_LEVELS`, it **must** be non-zero and VkImageSubresourceRange::`baseMipLevel` + VkImageSubresourceRange::`levelCount` for that element of the `pRanges` array **must** be less than or equal to the `mipLevels` specified in VkImageCreateInfo when `image` was created
- 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
- If the VkImageSubresourceRange::`layerCount` member of any element of the `pRanges` array is not `VK_REMAINING_ARRAY_LAYERS`, it **must** be non-zero and VkImageSubresourceRange::`baseArrayLayer` + VkImageSubresourceRange::`layerCount` for that element of the `pRanges` array **must** be less than or equal to the `arrayLayers` specified in VkImageCreateInfo when `image` was created
- `image` **must** have a depth/stencil format

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `image` **must** be a valid `VkImage` handle
- `imageLayout` **must** be a valid VkImageLayout value
- `pDepthStencil` **must** be a pointer to a valid `VkClearDepthStencilValue` structure
- `pRanges` **must** be a pointer to an array of `rangeCount` valid `VkImageSubresourceRange` structures
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- This command **must** only be called outside of a render pass instance
- `rangeCount` **must** be greater than `0`
- 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	Pipeline Type
Primary Secondary	Outside	Graphics	Transfer

See Also

VkClearDepthStencilValue, VkCommandBuffer, VkImage, VkImageLayout, VkImageSubresourceRange

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdClearDepthStencilImage

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdCopyBuffer(3)

Name

vkCmdCopyBuffer - Copy data between buffer regions

C Specification

To copy data between buffer objects, call:

```
void vkCmdCopyBuffer(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    srcBuffer,
    VkBuffer                                    dstBuffer,
    uint32_t                                    regionCount,
    const VkBufferCopy*                         pRegions);
```

Parameters

- `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.

Description

Each region in `pRegions` is copied from the source buffer to the same region of the destination buffer. `srcBuffer` and `dstBuffer` **can** be the same buffer or alias the same memory, but the result is undefined if the copy regions overlap in memory.

Valid Usage

- The `size` member of a given element of `pRegions` **must** be greater than `0`
- The `srcOffset` member of a given element of `pRegions` **must** be less than the size of `srcBuffer`
- The `dstOffset` member of a given element of `pRegions` **must** be less than the size of `dstBuffer`
- The `size` member of a given element of `pRegions` **must** be less than or equal to the size of `srcBuffer` minus `srcOffset`
- The `size` member of a given element of `pRegions` **must** be less than or equal to the size of `dstBuffer` minus `dstOffset`
- The union of the source regions, and the union of the destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- `srcBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_SRC_BIT` usage flag
- If `srcBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- `dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `srcBuffer` **must** be a valid `VkBuffer` handle
- `dstBuffer` **must** be a valid `VkBuffer` handle
- `pRegions` **must** be a pointer to an array of `regionCount` `VkBufferCopy` structures
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- This command **must** only be called outside of a render pass instance
- `regionCount` **must** be greater than `0`
- 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	Pipeline Type
Primary Secondary	Outside	Transfer graphics compute	Transfer

See Also

VkBuffer, VkBufferCopy, VkCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdCopyBuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdCopyBufferToImage(3)

Name

vkCmdCopyBufferToImage - Copy data from a buffer into an image

C Specification

To copy data from a buffer object to an image object, call:

```
void vkCmdCopyBufferToImage(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    srcBuffer,
    VkImage                                     dstImage,
    VkImageLayout                               dstImageLayout,
    uint32_t                                    regionCount,
    const VkBufferImageCopy*                    pRegions);
```

Parameters

- `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.

Description

Each region in `pRegions` is copied from the specified region of the source buffer to the specified region of the destination image.

Valid Usage

- The buffer region specified by a given element of `pRegions` **must** be a region that is contained within `srcBuffer`
- The image region specified by a given element of `pRegions` **must** be a region that is contained within `dstImage`
- The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- `srcBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_SRC_BIT` usage flag
- If `srcBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- `dstImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag
- If `dstImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- `dstImage` **must** have a sample count equal to `VK_SAMPLE_COUNT_1_BIT`
- `dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `srcBuffer` **must** be a valid `VkBuffer` handle
- `dstImage` **must** be a valid `VkImage` handle
- `dstImageLayout` **must** be a valid VkImageLayout value
- `pRegions` **must** be a pointer to an array of `regionCount` valid `VkBufferImageCopy` structures
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- This command **must** only be called outside of a render pass instance
- `regionCount` **must** be greater than `0`
- 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	Pipeline Type
Primary Secondary	Outside	Transfer graphics compute	Transfer

See Also

VkBuffer, VkBufferImageCopy, VkCommandBuffer, VkImage, VkImageLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdCopyBufferToImage

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdCopyImage(3)

Name

vkCmdCopyImage - Copy data between images

C Specification

To copy data between image objects, call:

```
void vkCmdCopyImage(
    VkCommandBuffer                             commandBuffer,
    VkImage                                     srcImage,
    VkImageLayout                               srcImageLayout,
    VkImage                                     dstImage,
    VkImageLayout                               dstImageLayout,
    uint32_t                                    regionCount,
    const VkImageCopy*                          pRegions);
```

Parameters

- `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.

Description

Each region in `pRegions` is copied from the source image to the same region of the destination image. `srcImage` and `dstImage` **can** be the same image or alias the same memory.

The formats of `srcImage` and `dstImage` **must** be compatible. Formats are considered compatible if their element size is the same between both formats. For example, `VK_FORMAT_R8G8B8A8_UNORM` is compatible with `VK_FORMAT_R32_UINT` because both texels are 4 bytes in size. Depth/stencil formats **must** match exactly.

`vkCmdCopyImage` allows copying between size-compatible compressed and uncompressed internal formats. Formats are size-compatible if the element size of the uncompressed format is equal to the element size (compressed texel block size) of the compressed format. Such a copy does not perform on-the-fly compression or decompression. When copying from an uncompressed format to a compressed format, each texel of uncompressed data of the source image is copied as a raw value to the corresponding compressed texel block of the destination image. When copying from a compressed format to an uncompressed format, each compressed texel block of the source image is copied as a raw value to the corresponding texel of uncompressed data in the destination image. Thus, for example, it is legal to copy between a 128-bit uncompressed format and a compressed format which has a 128-bit sized compressed texel block representing 4×4 texels (using 8 bits per texel), or between a 64-bit uncompressed format and a compressed format which has a 64-bit sized compressed texel block representing 4×4 texels (using 4 bits per texel).

When copying between compressed and uncompressed formats the `extent` members represent the texel dimensions of the source image and not the destination. When copying from a compressed image to an uncompressed image the image texel dimensions written to the uncompressed image will be source extent divided by the compressed texel block dimensions. When copying from an uncompressed image to a compressed image the image texel dimensions written to the compressed image will be the source extent multiplied by the compressed texel block dimensions. In both cases the number of bytes read and the number of bytes written will be identical.

Copying to or from block-compressed images is typically done in multiples of the compressed texel block size. For this reason the `extent` **must** be a multiple of the compressed texel block dimension. There is one exception to this rule which is **required** to handle compressed images created with dimensions that are not a multiple of the compressed texel block dimensions: if the `srcImage` is compressed, then:

- If `extent.width` is not a multiple of the compressed texel block width, then (`extent.width` + `srcOffset.x`) **must** equal the image subresource width.
- If `extent.height` is not a multiple of the compressed texel block height, then (`extent.height` + `srcOffset.y`) **must** equal the image subresource height.
- If `extent.depth` is not a multiple of the compressed texel block depth, then (`extent.depth` + `srcOffset.z`) **must** equal the image subresource depth.

Similarly, if the `dstImage` is compressed, then:

- If `extent.width` is not a multiple of the compressed texel block width, then (`extent.width` + `dstOffset.x`) **must** equal the image subresource width.
- If `extent.height` is not a multiple of the compressed texel block height, then (`extent.height` + `dstOffset.y`) **must** equal the image subresource height.
- If `extent.depth` is not a multiple of the compressed texel block depth, then (`extent.depth` + `dstOffset.z`) **must** equal the image subresource depth.

This allows the last compressed texel block of the image in each non-multiple dimension to be included as a source or destination of the copy.

`vkCmdCopyImage` **can** be used to copy image data between multisample images, but both images **must** have the same number of samples.

Valid Usage

- The source region specified by a given element of `pRegions` **must** be a region that is contained within `srcImage`
- The destination region specified by a given element of `pRegions` **must** be a region that is contained within `dstImage`
- The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- `srcImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` usage flag
- If `srcImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- `srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- `srcImageLayout` **must** be `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- `dstImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` usage flag
- If `dstImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- `dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- The VkFormat of each of `srcImage` and `dstImage` **must** be compatible, as defined below
- The sample count of `srcImage` and `dstImage` **must** match

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `srcImage` **must** be a valid `VkImage` handle
- `srcImageLayout` **must** be a valid VkImageLayout value
- `dstImage` **must** be a valid `VkImage` handle
- `dstImageLayout` **must** be a valid VkImageLayout value
- `pRegions` **must** be a pointer to an array of `regionCount` valid `VkImageCopy` structures
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- This command **must** only be called outside of a render pass instance
- `regionCount` **must** be greater than `0`
- 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	Pipeline Type
Primary Secondary	Outside	Transfer graphics compute	Transfer

See Also

VkCommandBuffer, VkImage, VkImageCopy, VkImageLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdCopyImage

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdCopyImageToBuffer(3)

Name

vkCmdCopyImageToBuffer - Copy image data into a buffer

C Specification

To copy data from an image object to a buffer object, call:

```
void vkCmdCopyImageToBuffer(
    VkCommandBuffer                             commandBuffer,
    VkImage                                     srcImage,
    VkImageLayout                               srcImageLayout,
    VkBuffer                                    dstBuffer,
    uint32_t                                    regionCount,
    const VkBufferImageCopy*                    pRegions);
```

Parameters

- `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.

Description

Each region in `pRegions` is copied from the specified region of the source image to the specified region of the destination buffer.

Valid Usage

- The image region specified by a given element of `pRegions` **must** be a region that is contained within `srcImage`
- The buffer region specified by a given element of `pRegions` **must** be a region that is contained within `dstBuffer`
- The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- `srcImage` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` usage flag
- If `srcImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- `srcImage` **must** have a sample count equal to `VK_SAMPLE_COUNT_1_BIT`
- `srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- `srcImageLayout` **must** be `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- `dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `srcImage` **must** be a valid `VkImage` handle
- `srcImageLayout` **must** be a valid VkImageLayout value
- `dstBuffer` **must** be a valid `VkBuffer` handle
- `pRegions` **must** be a pointer to an array of `regionCount` valid `VkBufferImageCopy` structures
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- This command **must** only be called outside of a render pass instance
- `regionCount` **must** be greater than `0`
- 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	Pipeline Type
Primary Secondary	Outside	Transfer graphics compute	Transfer

See Also

VkBuffer, VkBufferImageCopy, VkCommandBuffer, VkImage, VkImageLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdCopyImageToBuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdCopyQueryPoolResults(3)

Name

vkCmdCopyQueryPoolResults - Copy the results of queries in a query pool to a buffer object

C Specification

To copy query statuses and numerical results directly to buffer memory, call:

```
void vkCmdCopyQueryPoolResults(
    VkCommandBuffer                             commandBuffer,
    VkQueryPool                                 queryPool,
    uint32_t                                    firstQuery,
    uint32_t                                    queryCount,
    VkBuffer                                    dstBuffer,
    VkDeviceSize                                dstOffset,
    VkDeviceSize                                stride,
    VkQueryResultFlags                          flags);
```

Parameters

- `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.

Description

`vkCmdCopyQueryPoolResults` is guaranteed to see the effect of previous uses of `vkCmdResetQueryPool` in the same queue, without any additional synchronization. Thus, the results will always reflect the most recent use of the query.

`flags` has the same possible values described above for the `flags` parameter of vkGetQueryPoolResults, but the different style of execution causes some subtle behavioral differences. Because `vkCmdCopyQueryPoolResults` executes in order with respect to other query commands, there is less ambiguity about which use of a query is being requested.

If no bits are set in flags, results for all requested queries in the available state are written as 32-bit unsigned integer values, and nothing is written for queries in the unavailable state.

If VK_QUERY_RESULT_64_BIT is set, the results are written as an array of 64-bit unsigned integer values as described for vkGetQueryPoolResults.

If VK_QUERY_RESULT_WAIT_BIT is set, the implementation will wait for each query's status to be in the available state before retrieving the numerical results for that query. This is guaranteed to reflect the most recent use of the query on the same queue, assuming that the query is not being simultaneously used by other queues. If the query does not become available in a finite amount of time (e.g. due to not issuing a query since the last reset), a VK_ERROR_DEVICE_LOST error **may** occur.

Similarly, if VK_QUERY_RESULT_WITH_AVAILABILITY_BIT is set and VK_QUERY_RESULT_WAIT_BIT is not set, the availability is guaranteed to reflect the most recent use of the query on the same queue, assuming that the query is not being simultaneously used by other queues. As with vkGetQueryPoolResults, implementations **must** guarantee that if they return a non-zero availability value, then the numerical results are valid.

If VK_QUERY_RESULT_PARTIAL_BIT is set, VK_QUERY_RESULT_WAIT_BIT is not set, and the query's status is unavailable, an intermediate result value between zero and the final result value is written for that query.

VK_QUERY_RESULT_PARTIAL_BIT **must** not be used if the pool's queryType is VK_QUERY_TYPE_TIMESTAMP.

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

- dstOffset **must** be less than the size of dstBuffer
- firstQuery **must** be less than the number of queries in queryPool
- The sum of firstQuery and queryCount **must** be less than or equal to the number of queries in queryPool
- If VK_QUERY_RESULT_64_BIT is not set in flags then dstOffset and stride **must** be multiples of 4
- If VK_QUERY_RESULT_64_BIT is set in flags then dstOffset and stride **must** be multiples of 8
- dstBuffer **must** have enough storage, from dstOffset, to contain the result of each query, as described here
- dstBuffer **must** have been created with VK_BUFFER_USAGE_TRANSFER_DST_BIT usage flag
- If dstBuffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- If the queryType used to create queryPool was VK_QUERY_TYPE_TIMESTAMP, flags **must** not contain VK_QUERY_RESULT_PARTIAL_BIT

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `queryPool` **must** be a valid `VkQueryPool` handle
- `dstBuffer` **must** be a valid `VkBuffer` handle
- `flags` **must** be a valid combination of VkQueryResultFlagBits values
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- This command **must** only be called outside of a render pass instance
- 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	Pipeline Type
Primary Secondary	Outside	Graphics compute	Transfer

See Also

VkBuffer, VkCommandBuffer, `VkDeviceSize`, VkQueryPool, VkQueryResultFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdCopyQueryPoolResults

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdDispatch(3)

Name

vkCmdDispatch - Dispatch compute work items

C Specification

To record a dispatch, call:

```
void vkCmdDispatch(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    groupCountX,
    uint32_t                                    groupCountY,
    uint32_t                                    groupCountZ);
```

Parameters

- `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.

Description

When the command is executed, a global workgroup consisting of groupCountX × groupCountY × groupCountZ local workgroups is assembled.

Valid Usage

- `groupCountX` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[0]
- `groupCountY` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[1]
- `groupCountZ` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[2]
- For each set *n* that is statically used by the `VkPipeline` currently bound to `VK_PIPELINE_BIND_POINT_COMPUTE`, a descriptor set **must** have been bound to *n* at `VK_PIPELINE_BIND_POINT_COMPUTE`, with a `VkPipelineLayout` that is compatible for set *n*, with the `VkPipelineLayout` used to create the current `VkPipeline`, as described in html/vkspec.html#descriptorsets-compatibility
- Descriptors in each bound descriptor set, specified via `vkCmdBindDescriptorSets`, **must** be valid if they are statically used by the currently bound `VkPipeline` object, specified via `vkCmdBindPipeline`
- A valid compute pipeline **must** be bound to the current command buffer with `VK_PIPELINE_BIND_POINT_COMPUTE`
- For each push constant that is statically used by the `VkPipeline` currently bound to `VK_PIPELINE_BIND_POINT_COMPUTE`, a push constant value **must** have been set for `VK_PIPELINE_BIND_POINT_COMPUTE`, with a `VkPipelineLayout` that is compatible for push constants with the one used to create the current `VkPipeline`, as described in html/vkspec.html#descriptorsets-compatibility
- If any `VkSampler` object that is accessed from a shader by the `VkPipeline` currently bound to `VK_PIPELINE_BIND_POINT_COMPUTE` uses unnormalized coordinates, it **must** not be used to sample from any `VkImage` with a `VkImageView` of the type `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY` or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`, in any shader stage
- If any `VkSampler` object that is accessed from a shader by the `VkPipeline` currently bound to `VK_PIPELINE_BIND_POINT_COMPUTE` uses unnormalized coordinates, it **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions with `ImplicitLod`, `Dref` or `Proj` in their name, in any shader stage
- If any `VkSampler` object that is accessed from a shader by the `VkPipeline` currently bound to `VK_PIPELINE_BIND_POINT_COMPUTE` uses unnormalized coordinates, it **must** not be used with any of the SPIR-V `OpImageSample*` or `OpImageSparseSample*` instructions that includes a LOD bias or any offset values, in any shader stage
- If the robust buffer access feature is not enabled, and any shader stage in the `VkPipeline` object currently bound to `VK_PIPELINE_BIND_POINT_COMPUTE` accesses a uniform buffer, it **must** not access values outside of the range of that buffer specified in the currently bound descriptor set
- If the robust buffer access feature is not enabled, and any shader stage in the `VkPipeline` object currently bound to `VK_PIPELINE_BIND_POINT_COMPUTE` accesses a storage buffer, it **must** not access values outside of the range of that buffer specified in the currently bound

descriptor set

- Any `VkImageView` being sampled with `VK_FILTER_LINEAR` as a result of this command **must** be of a format which supports linear filtering, as specified by the `VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT` flag in `VkFormatProperties::linearTilingFeatures` (for a linear image) or `VkFormatProperties::optimalTilingFeatures`(for an optimally tiled image) returned by `vkGetPhysicalDeviceFormatProperties`

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support compute operations
- 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	Pipeline Type
Primary Secondary	Outside	Compute	Compute

See Also

VkCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdDispatch

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdDispatchIndirect(3)

Name

vkCmdDispatchIndirect - Dispatch compute work items using indirect parameters

C Specification

To record an indirect command dispatch, call:

```
void vkCmdDispatchIndirect(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset);
```

Parameters

- `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.

Description

`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

- If buffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- For each set n that is statically used by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_COMPUTE, a descriptor set **must** have been bound to n at VK_PIPELINE_BIND_POINT_COMPUTE, with a VkPipelineLayout that is compatible for set n, with the VkPipelineLayout used to create the current VkPipeline, as described in html/vkspec.html#descriptorsets-compatibility
- Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid if they are statically used by the currently bound VkPipeline object, specified via vkCmdBindPipeline
- A valid compute pipeline **must** be bound to the current command buffer with VK_PIPELINE_BIND_POINT_COMPUTE
- buffer **must** have been created with the VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT bit set
- offset **must** be a multiple of 4
- The sum of offset and the size of VkDispatchIndirectCommand **must** be less than or equal to the size of buffer
- For each push constant that is statically used by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_COMPUTE, a push constant value **must** have been set for VK_PIPELINE_BIND_POINT_COMPUTE, with a VkPipelineLayout that is compatible for push constants with the one used to create the current VkPipeline, as described in html/vkspec.html#descriptorsets-compatibility
- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_COMPUTE uses unnormalized coordinates, it **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_COMPUTE uses unnormalized coordinates, it **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_COMPUTE uses unnormalized coordinates, it **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- If the robust buffer access feature is not enabled, and any shader stage in the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_COMPUTE accesses a uniform buffer, it **must** not access values outside of the range of that buffer specified in the currently bound descriptor set
- If the robust buffer access feature is not enabled, and any shader stage in the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_COMPUTE accesses a storage buffer, it **must** not access values outside of the range of that buffer specified in the currently bound

descriptor set

- Any VkImageView being sampled with VK_FILTER_LINEAR as a result of this command **must** be of a format which supports linear filtering, as specified by the VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT flag in VkFormatProperties ::linearTilingFeatures (for a linear image) or VkFormatProperties:: optimalTilingFeatures(for an optimally tiled image) returned by vkGetPhysicalDeviceFormatProperties

Valid Usage (Implicit)

- commandBuffer **must** be a valid VkCommandBuffer handle
- buffer **must** be a valid VkBuffer handle
- commandBuffer **must** be in the recording state
- The VkCommandPool that commandBuffer was allocated from **must** support compute operations
- This command **must** only be called outside of a render pass instance
- 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	Pipeline Type
Primary Secondary	Outside	Compute	Compute

See Also

VkBuffer, VkCommandBuffer, VkDeviceSize

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdDispatchIndirect

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdDraw(3)

Name

vkCmdDraw - Draw primitives

C Specification

To record a non-indexed draw, call:

```
void vkCmdDraw(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    vertexCount,
    uint32_t                                    instanceCount,
    uint32_t                                    firstVertex,
    uint32_t                                    firstInstance);
```

Parameters

- `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.

Description

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 currently bound graphics pipeline.

Valid Usage

- The current render pass **must** be compatible with the renderPass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS.
- The subpass index of the current render pass **must** be equal to the subpass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS.
- For each set *n* that is statically used by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS, a descriptor set **must** have been bound to *n* at VK_PIPELINE_BIND_POINT_GRAPHICS, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline, as described in html/vkspec.html#descriptorsets-compatibility
- For each push constant that is statically used by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS, a push constant value **must** have been set for VK_PIPELINE_BIND_POINT_GRAPHICS, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline, as described in html/vkspec.html#descriptorsets-compatibility
- Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid if they are statically used by the currently bound VkPipeline object, specified via vkCmdBindPipeline
- All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have valid buffers bound
- For a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in html/vkspec.html#fxvertex-input
- A valid graphics pipeline **must** be bound to the current command buffer with VK_PIPELINE_BIND_POINT_GRAPHICS
- If the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS requires any dynamic state, that state **must** have been set on the current command buffer
- Every input attachment used by the current subpass **must** be bound to the pipeline via a descriptor set
- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS uses unnormalized coordinates, it **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS uses unnormalized coordinates, it **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS uses unnormalized coordinates, it **must** not be used with

any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage

- If the robust buffer access feature is not enabled, and any shader stage in the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS accesses a uniform buffer, it **must** not access values outside of the range of that buffer specified in the currently bound descriptor set
- If the robust buffer access feature is not enabled, and any shader stage in the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS accesses a storage buffer, it **must** not access values outside of the range of that buffer specified in the currently bound descriptor set
- Any VkImageView being sampled with VK_FILTER_LINEAR as a result of this command **must** be of a format which supports linear filtering, as specified by the VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT flag in VkFormatProperties::linearTilingFeatures (for a linear image) or VkFormatProperties::optimalTilingFeatures(for an optimally tiled image) returned by vkGetPhysicalDeviceFormatProperties

Valid Usage (Implicit)

- commandBuffer **must** be a valid VkCommandBuffer handle
- commandBuffer **must** be in the recording state
- The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- 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	Pipeline Type
Primary Secondary	Inside	Graphics	Graphics

See Also

VkCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdDraw

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdDrawIndexed(3)

Name

vkCmdDrawIndexed - Issue an indexed draw into a command buffer

C Specification

To record an indexed draw, call:

```
void vkCmdDrawIndexed(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    indexCount,
    uint32_t                                    instanceCount,
    uint32_t                                    firstIndex,
    int32_t                                     vertexOffset,
    uint32_t                                    firstInstance);
```

Parameters

- `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.

Description

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 `firstIndex` * `indexSize` + `offset` within the currently 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_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 currently bound graphics pipeline.

Valid Usage

- The current render pass **must** be compatible with the `renderPass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` currently bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`.
- The subpass index of the current render pass **must** be equal to the `subpass` member of the `VkGraphicsPipelineCreateInfo` structure specified when creating the `VkPipeline` currently bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`.
- For each set *n* that is statically used by the `VkPipeline` currently bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`, a descriptor set **must** have been bound to *n* at `VK_PIPELINE_BIND_POINT_GRAPHICS`, with a `VkPipelineLayout` that is compatible for set *n*, with the `VkPipelineLayout` used to create the current `VkPipeline`, as described in html/vkspec.html#descriptorsets-compatibility
- For each push constant that is statically used by the `VkPipeline` currently bound to `VK_PIPELINE_BIND_POINT_GRAPHICS`, a push constant value **must** have been set for `VK_PIPELINE_BIND_POINT_GRAPHICS`, with a `VkPipelineLayout` that is compatible for push constants, with the `VkPipelineLayout` used to create the current `VkPipeline`, as described in html/vkspec.html#descriptorsets-compatibility
- Descriptors in each bound descriptor set, specified via `vkCmdBindDescriptorSets`, **must** be valid if they are statically used by the currently bound `VkPipeline` object, specified via `vkCmdBindPipeline`
- All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have valid buffers bound
- For a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in html/vkspec.html#fxvertex-input
- A valid graphics pipeline **must** be bound to the current command buffer with `VK_PIPELINE_BIND_POINT_GRAPHICS`
- If the `VkPipeline` object currently bound to `VK_PIPELINE_BIND_POINT_GRAPHICS` requires any dynamic state, that state **must** have been set on the current command buffer
- (`indexSize` * (`firstIndex` + `indexCount`) + `offset`) **must** be less than or equal to the size of the currently bound index buffer, with indexSize being based on the type specified by `indexType`, where the index buffer, `indexType`, and `offset` are specified via `vkCmdBindIndexBuffer`
- Every input attachment used by the current subpass **must** be bound to the pipeline via a descriptor set
- If any `VkSampler` object that is accessed from a shader by the `VkPipeline` currently bound to `VK_PIPELINE_BIND_POINT_GRAPHICS` uses unnormalized coordinates, it **must** not be used to sample from any `VkImage` with a `VkImageView` of the type `VK_IMAGE_VIEW_TYPE_3D`, `VK_IMAGE_VIEW_TYPE_CUBE`, `VK_IMAGE_VIEW_TYPE_1D_ARRAY`, `VK_IMAGE_VIEW_TYPE_2D_ARRAY` or `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`, in any shader stage
- If any `VkSampler` object that is accessed from a shader by the `VkPipeline` currently bound to `VK_PIPELINE_BIND_POINT_GRAPHICS` uses unnormalized coordinates, it **must** not be used with

any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage

- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS uses unnormalized coordinates, it **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- If the robust buffer access feature is not enabled, and any shader stage in the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS accesses a uniform buffer, it **must** not access values outside of the range of that buffer specified in the currently bound descriptor set
- If the robust buffer access feature is not enabled, and any shader stage in the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS accesses a storage buffer, it **must** not access values outside of the range of that buffer specified in the currently bound descriptor set
- Any VkImageView being sampled with VK_FILTER_LINEAR as a result of this command **must** be of a format which supports linear filtering, as specified by the VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT flag in VkFormatProperties::linearTilingFeatures (for a linear image) or VkFormatProperties::optimalTilingFeatures(for an optimally tiled image) returned by vkGetPhysicalDeviceFormatProperties

Valid Usage (Implicit)

- commandBuffer **must** be a valid VkCommandBuffer handle
- commandBuffer **must** be in the recording state
- The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- 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	Pipeline Type
Primary Secondary	Inside	Graphics	Graphics

See Also

VkCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdDrawIndexed

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdDrawIndexedIndirect(3)

Name

vkCmdDrawIndexedIndirect - Perform an indexed indirect draw

C Specification

To record an indexed indirect draw, call:

```
void vkCmdDrawIndexedIndirect(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    uint32_t                                    drawCount,
    uint32_t                                    stride);
```

Parameters

- `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.

Description

`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

- If buffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- offset **must** be a multiple of 4
- If drawCount is greater than 1, stride **must** be a multiple of 4 and **must** be greater than or equal to sizeof(VkDrawIndexedIndirectCommand)
- If the multi-draw indirect feature is not enabled, drawCount **must** be 0 or 1
- If the drawIndirectFirstInstance feature is not enabled, all the firstInstance members of the VkDrawIndexedIndirectCommand structures accessed by this command **must** be 0
- The current render pass **must** be compatible with the renderPass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS.
- The subpass index of the current render pass **must** be equal to the subpass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS.
- For each set *n* that is statically used by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS, a descriptor set **must** have been bound to *n* at VK_PIPELINE_BIND_POINT_GRAPHICS, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline, as described in html/vkspec.html#descriptorsets-compatibility
- For each push constant that is statically used by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS, a push constant value **must** have been set for VK_PIPELINE_BIND_POINT_GRAPHICS, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline, as described in html/vkspec.html#descriptorsets-compatibility
- Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid if they are statically used by the currently bound VkPipeline object, specified via vkCmdBindPipeline
- All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have valid buffers bound
- A valid graphics pipeline **must** be bound to the current command buffer with VK_PIPELINE_BIND_POINT_GRAPHICS
- If the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS requires any dynamic state, that state **must** have been set on the current command buffer
- If drawCount is equal to 1, (offset + sizeof(VkDrawIndexedIndirectCommand)) **must** be less than or equal to the size of buffer
- If drawCount is greater than 1, (stride × (drawCount - 1) + offset + sizeof(VkDrawIndexedIndirectCommand)) **must** be less than or equal to the size of buffer
- drawCount **must** be less than or equal to VkPhysicalDeviceLimits::maxDrawIndirectCount
- Every input attachment used by the current subpass **must** be bound to the pipeline via a

descriptor set

- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS uses unnormalized coordinates, it **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS uses unnormalized coordinates, it **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS uses unnormalized coordinates, it **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- If the robust buffer access feature is not enabled, and any shader stage in the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS accesses a uniform buffer, it **must** not access values outside of the range of that buffer specified in the currently bound descriptor set
- If the robust buffer access feature is not enabled, and any shader stage in the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS accesses a storage buffer, it **must** not access values outside of the range of that buffer specified in the currently bound descriptor set
- Any VkImageView being sampled with VK_FILTER_LINEAR as a result of this command **must** be of a format which supports linear filtering, as specified by the VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT flag in VkFormatProperties ::linearTilingFeatures (for a linear image) or VkFormatProperties::optimalTilingFeatures(for an optimally tiled image) returned by vkGetPhysicalDeviceFormatProperties

Valid Usage (Implicit)

- commandBuffer **must** be a valid VkCommandBuffer handle
- buffer **must** be a valid VkBuffer handle
- commandBuffer **must** be in the recording state
- The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- This command **must** only be called inside of a render pass instance
- 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	Pipeline Type
Primary Secondary	Inside	Graphics	Graphics

See Also

VkBuffer, VkCommandBuffer, `VkDeviceSize`

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdDrawIndexedIndirect

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdDrawIndirect(3)

Name

vkCmdDrawIndirect - Issue an indirect draw into a command buffer

C Specification

To record a non-indexed indirect draw, call:

```
void vkCmdDrawIndirect(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    buffer,
    VkDeviceSize                                offset,
    uint32_t                                    drawCount,
    uint32_t                                    stride);
```

Parameters

- `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.

Description

`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

- If buffer is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- offset **must** be a multiple of 4
- If drawCount is greater than 1, stride **must** be a multiple of 4 and **must** be greater than or equal to sizeof(VkDrawIndirectCommand)
- If the multi-draw indirect feature is not enabled, drawCount **must** be 0 or 1
- If the drawIndirectFirstInstance feature is not enabled, all the firstInstance members of the VkDrawIndirectCommand structures accessed by this command **must** be 0
- The current render pass **must** be compatible with the renderPass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS.
- The subpass index of the current render pass **must** be equal to the subpass member of the VkGraphicsPipelineCreateInfo structure specified when creating the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS.
- For each set *n* that is statically used by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS, a descriptor set **must** have been bound to *n* at VK_PIPELINE_BIND_POINT_GRAPHICS, with a VkPipelineLayout that is compatible for set *n*, with the VkPipelineLayout used to create the current VkPipeline, as described in html/vkspec.html#descriptorsets-compatibility
- For each push constant that is statically used by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS, a push constant value **must** have been set for VK_PIPELINE_BIND_POINT_GRAPHICS, with a VkPipelineLayout that is compatible for push constants, with the VkPipelineLayout used to create the current VkPipeline, as described in html/vkspec.html#descriptorsets-compatibility
- Descriptors in each bound descriptor set, specified via vkCmdBindDescriptorSets, **must** be valid if they are statically used by the currently bound VkPipeline object, specified via vkCmdBindPipeline
- All vertex input bindings accessed via vertex input variables declared in the vertex shader entry point's interface **must** have valid buffers bound
- A valid graphics pipeline **must** be bound to the current command buffer with VK_PIPELINE_BIND_POINT_GRAPHICS
- If the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS requires any dynamic state, that state **must** have been set on the current command buffer
- If drawCount is equal to 1, (offset + sizeof(VkDrawIndirectCommand)) **must** be less than or equal to the size of buffer
- If drawCount is greater than 1, (stride × (drawCount - 1) + offset + sizeof(VkDrawIndirectCommand)) **must** be less than or equal to the size of buffer
- drawCount **must** be less than or equal to VkPhysicalDeviceLimits::maxDrawIndirectCount
- Every input attachment used by the current subpass **must** be bound to the pipeline via a

descriptor set

- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS uses unnormalized coordinates, it **must** not be used to sample from any VkImage with a VkImageView of the type VK_IMAGE_VIEW_TYPE_3D, VK_IMAGE_VIEW_TYPE_CUBE, VK_IMAGE_VIEW_TYPE_1D_ARRAY, VK_IMAGE_VIEW_TYPE_2D_ARRAY or VK_IMAGE_VIEW_TYPE_CUBE_ARRAY, in any shader stage
- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS uses unnormalized coordinates, it **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions with ImplicitLod, Dref or Proj in their name, in any shader stage
- If any VkSampler object that is accessed from a shader by the VkPipeline currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS uses unnormalized coordinates, it **must** not be used with any of the SPIR-V OpImageSample* or OpImageSparseSample* instructions that includes a LOD bias or any offset values, in any shader stage
- If the robust buffer access feature is not enabled, and any shader stage in the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS accesses a uniform buffer, it **must** not access values outside of the range of that buffer specified in the currently bound descriptor set
- If the robust buffer access feature is not enabled, and any shader stage in the VkPipeline object currently bound to VK_PIPELINE_BIND_POINT_GRAPHICS accesses a storage buffer, it **must** not access values outside of the range of that buffer specified in the currently bound descriptor set
- Any VkImageView being sampled with VK_FILTER_LINEAR as a result of this command **must** be of a format which supports linear filtering, as specified by the VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT flag in VkFormatProperties::linearTilingFeatures (for a linear image) or VkFormatProperties::optimalTilingFeatures(for an optimally tiled image) returned by vkGetPhysicalDeviceFormatProperties

Valid Usage (Implicit)

- commandBuffer **must** be a valid VkCommandBuffer handle
- buffer **must** be a valid VkBuffer handle
- commandBuffer **must** be in the recording state
- The VkCommandPool that commandBuffer was allocated from **must** support graphics operations
- This command **must** only be called inside of a render pass instance
- 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	Pipeline Type
Primary Secondary	Inside	Graphics	Graphics

See Also

VkBuffer, VkCommandBuffer, `VkDeviceSize`

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdDrawIndirect

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdEndQuery(3)

Name

vkCmdEndQuery - Ends a query

C Specification

To end a query after the set of desired draw or dispatch commands is executed, call:

```
void vkCmdEndQuery(
    VkCommandBuffer                             commandBuffer,
    VkQueryPool                                 queryPool,
    uint32_t                                    query);
```

Parameters

- `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.

Description

As queries operate asynchronously, ending a query does not immediately set the query's status to available. A query is considered *finished* when the final results of the query are ready to be retrieved by vkGetQueryPoolResults and vkCmdCopyQueryPoolResults, and this is when the query's status is set to available.

Once a query is ended the query **must** finish in finite time, unless the state of the query is changed using other commands, e.g. by issuing a reset of the query.

Valid Usage

- The query identified by `queryPool` and `query` **must** currently be active
- `query` **must** be less than the number of queries in `queryPool`

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `queryPool` **must** be a valid `VkQueryPool` handle
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- 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	Pipeline Type
Primary Secondary	Both	Graphics compute	

See Also

VkCommandBuffer, VkQueryPool

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdEndQuery

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdEndRenderPass(3)

Name

vkCmdEndRenderPass - End the current render pass

C Specification

To record a command to end a render pass instance after recording the commands for the last subpass, call:

```
void vkCmdEndRenderPass(
    VkCommandBuffer                             commandBuffer);
```

Parameters

- `commandBuffer` is the command buffer in which to end the current render pass instance.

Description

Ending a render pass instance performs any multisample resolve operations on the final subpass.

Valid Usage

- The current subpass index **must** be equal to the number of subpasses in the render pass minus one

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- This command **must** only be called inside of a render pass instance
- `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	Pipeline Type
Primary	Inside	Graphics	Graphics

See Also

VkCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdEndRenderPass

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdExecuteCommands(3)

Name

vkCmdExecuteCommands - Execute a secondary command buffer from a primary command buffer

C Specification

A secondary command buffer **must** not be directly submitted to a queue. Instead, secondary command buffers are recorded to execute as part of a primary command buffer with the command:

```
void vkCmdExecuteCommands(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    commandBufferCount,
    const VkCommandBuffer*                      pCommandBuffers);
```

Parameters

- `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 an array of secondary command buffer handles, which are recorded to execute in the primary command buffer in the order they are listed in the array.

Description

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

- commandBuffer **must** have been allocated with a level of VK_COMMAND_BUFFER_LEVEL_PRIMARY
- Any given element of pCommandBuffers **must** have been allocated with a level of VK_COMMAND_BUFFER_LEVEL_SECONDARY
- Any given element of pCommandBuffers **must** be in the pending or executable state.
- 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, that primary command buffer **must** not be in the pending state
- If any given element of pCommandBuffers was not recorded with the VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT flag, it **must** not be in the pending state.
- If any given 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.
- If any given 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.
- Any given 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
- If vkCmdExecuteCommands is being called within a render pass instance, that render pass instance **must** have been begun with the contents parameter of vkCmdBeginRenderPass set to VK_SUBPASS_CONTENTS_SECONDARY_COMMAND_BUFFERS
- If vkCmdExecuteCommands is being called within a render pass instance, any given element of pCommandBuffers **must** have been recorded with the VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT
- If vkCmdExecuteCommands is being called within a render pass instance, any given 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
- If vkCmdExecuteCommands is being called within a render pass instance, the render passes specified in the pname::pBeginInfo::pInheritanceInfo::renderPass members of the vkBeginCommandBuffer commands used to begin recording each element of pCommandBuffers **must** be compatible with the current render pass.
- If vkCmdExecuteCommands is being called within a render pass instance, and any given 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
- If vkCmdExecuteCommands is not being called within a render pass instance, any given element of pCommandBuffers **must** not have been recorded with the VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT
- If the inherited queries feature is not enabled, commandBuffer **must** not have any queries active

* 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`
* 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
* 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
* Any given element of `pCommandBuffers` **must** not begin any query types that are active in `commandBuffer`

Valid Usage (Implicit)

* `commandBuffer` **must** be a valid `VkCommandBuffer` handle
* `pCommandBuffers` **must** be a pointer to an array of `commandBufferCount` valid `VkCommandBuffer` handles
* `commandBuffer` **must** be in the recording state
* The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
* `commandBuffer` **must** be a primary `VkCommandBuffer`
* `commandBufferCount` **must** be greater than `0`
* 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	Pipeline Type
Primary	Both	Transfer graphics compute	

See Also

VkCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdExecuteCommands

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdFillBuffer(3)

Name

vkCmdFillBuffer - Fill a region of a buffer with a fixed value

C Specification

To clear buffer data, call:

```
void vkCmdFillBuffer(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    dstBuffer,
    VkDeviceSize                                dstOffset,
    VkDeviceSize                                size,
    uint32_t                                    data);
```

Parameters

- `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.

Description

`vkCmdFillBuffer` is treated as "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

- `dstOffset` **must** be less than the size of `dstBuffer`
- `dstOffset` **must** be a multiple of `4`
- If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be greater than `0`
- If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be less than or equal to the size of `dstBuffer` minus `dstOffset`
- If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be a multiple of `4`
- `dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics or compute operations
- If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `dstBuffer` **must** be a valid `VkBuffer` handle
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics or compute operations
- This command **must** only be called outside of a render pass instance
- 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	Pipeline Type
Primary Secondary	Outside	Graphics Compute	Transfer

See Also

VkBuffer, VkCommandBuffer, `VkDeviceSize`

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdFillBuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdNextSubpass(3)

Name

vkCmdNextSubpass - Transition to the next subpass of a render pass

C Specification

To transition to the next subpass in the render pass instance after recording the commands for a subpass, call:

```
void vkCmdNextSubpass(
    VkCommandBuffer                             commandBuffer,
    VkSubpassContents                           contents);
```

Parameters

- `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.

Description

The subpass index for a render pass begins at zero when `vkCmdBeginRenderPass` is recorded, and increments each time `vkCmdNextSubpass` is recorded.

Moving to the next subpass automatically performs any multisample resolve operations in the subpass being ended. End-of-subpass multisample resolves are treated as color attachment writes for the purposes of synchronization. That is, they are considered to execute in the `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT` pipeline stage and their writes are synchronized with `VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT`. Synchronization between rendering within a subpass and any resolve operations at the end of the subpass occurs automatically, without need for explicit dependencies or pipeline barriers. However, if the resolve attachment is also used in a different subpass, an explicit dependency is needed.

After transitioning to the next subpass, the application **can** record the commands for that subpass.

Valid Usage

- The current subpass index **must** be less than the number of subpasses in the render pass minus one

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `contents` **must** be a valid VkSubpassContents value
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- This command **must** only be called inside of a render pass instance
- `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	Pipeline Type
Primary	Inside	Graphics	Graphics

See Also

VkCommandBuffer, VkSubpassContents

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdNextSubpass

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdPipelineBarrier(3)

Name

vkCmdPipelineBarrier - Insert a memory dependency

C Specification

To record a pipeline barrier, call:

```
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);
```

Parameters

- `commandBuffer` is the command buffer into which the command is recorded.
- `srcStageMask` is a bitmask of VkPipelineStageFlagBits specifying the source stage mask.
- `dstStageMask` is a bitmask of VkPipelineStageFlagBits specifying the destination stage mask.
- `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.

Description

When vkCmdPipelineBarrier is submitted to a queue, it defines a memory dependency between commands that were submitted before it, and those submitted after it.

If vkCmdPipelineBarrier was recorded outside a render pass instance, the first synchronization scope includes every command submitted to the same queue before it, including those in the same command buffer and batch. If vkCmdPipelineBarrier was recorded inside a render pass instance, the first synchronization scope includes only commands submitted before it 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 every command submitted to the same queue after it, including those in the same command buffer and batch. If vkCmdPipelineBarrier was recorded inside a render pass instance, the second synchronization scope includes only commands submitted after it 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 access 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 access 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

- If the geometry shaders feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- If the geometry shaders feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- If the tessellation shaders feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- If the tessellation shaders feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- If vkCmdPipelineBarrier is called within a render pass instance, the render pass **must** have been created with a VkSubpassDependency instance in pDependencies that expresses a dependency from the current subpass to itself.
- If vkCmdPipelineBarrier is called within a render pass instance, srcStageMask **must** contain a subset of the bit values in the srcStageMask member of that instance of VkSubpassDependency
- If vkCmdPipelineBarrier is called within a render pass instance, dstStageMask **must** contain a subset of the bit values in the dstStageMask member of that instance of VkSubpassDependency
- If vkCmdPipelineBarrier is called within a render pass instance, the srcAccessMask of any element of pMemoryBarriers or pImageMemoryBarriers **must** contain a subset of the bit values the srcAccessMask member of that instance of VkSubpassDependency
- If vkCmdPipelineBarrier is called within a render pass instance, the dstAccessMask of any element of pMemoryBarriers or pImageMemoryBarriers **must** contain a subset of the bit values the dstAccessMask member of that instance of VkSubpassDependency
- If vkCmdPipelineBarrier is called within a render pass instance, dependencyFlags **must** be equal to the dependencyFlags member of that instance of VkSubpassDependency
- If vkCmdPipelineBarrier is called within a render pass instance, bufferMemoryBarrierCount **must** be 0
- If vkCmdPipelineBarrier is called within a render pass instance, the image member of any element of pImageMemoryBarriers **must** be equal to one of the elements of pAttachments that the current framebuffer was created with, that is also referred to by one of the elements of the pColorAttachments, pResolveAttachments or pDepthStencilAttachment members of the VkSubpassDescription instance that the current subpass was created with
- If vkCmdPipelineBarrier is called within a render pass instance, the oldLayout and newLayout members of any element of pImageMemoryBarriers **must** be equal to the layout member of an element of the pColorAttachments, pResolveAttachments or pDepthStencilAttachment members of the VkSubpassDescription instance that the current subpass was created with, that refers to the same image
- If vkCmdPipelineBarrier is called within a render pass instance, the oldLayout and newLayout members of an element of pImageMemoryBarriers **must** be equal

- If vkCmdPipelineBarrier is called within a render pass instance, the srcQueueFamilyIndex and dstQueueFamilyIndex members of any element of pImageMemoryBarriers **must** be VK_QUEUE_FAMILY_IGNORED
- Any pipeline stage included in srcStageMask or 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.
- Any given element of pMemoryBarriers, pBufferMemoryBarriers or pImageMemoryBarriers **must** not have any access flag included in its srcAccessMask member if that bit is not supported by any of the pipeline stages in srcStageMask, as specified in the table of supported access types.
- Any given element of pMemoryBarriers, pBufferMemoryBarriers or pImageMemoryBarriers **must** not have any access flag included in its dstAccessMask member if that bit is not supported by any of the pipeline stages in dstStageMask, as specified in the table of supported access types.

Valid Usage (Implicit)

- commandBuffer **must** be a valid VkCommandBuffer handle
- srcStageMask **must** be a valid combination of VkPipelineStageFlagBits values
- srcStageMask **must** not be 0
- dstStageMask **must** be a valid combination of VkPipelineStageFlagBits values
- dstStageMask **must** not be 0
- dependencyFlags **must** be a valid combination of VkDependencyFlagBits values
- If memoryBarrierCount is not 0, pMemoryBarriers **must** be a pointer to an array of memoryBarrierCount valid VkMemoryBarrier structures
- If bufferMemoryBarrierCount is not 0, pBufferMemoryBarriers **must** be a pointer to an array of bufferMemoryBarrierCount valid VkBufferMemoryBarrier structures
- If imageMemoryBarrierCount is not 0, pImageMemoryBarriers **must** be a pointer to an array of imageMemoryBarrierCount valid VkImageMemoryBarrier structures
- commandBuffer **must** be in the recording state
- The VkCommandPool that commandBuffer was allocated from **must** support transfer, graphics, or compute 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	Pipeline Type
Primary Secondary	Both	Transfer graphics compute	

See Also

VkBufferMemoryBarrier, VkCommandBuffer, VkDependencyFlags, VkImageMemoryBarrier, VkMemoryBarrier, VkPipelineStageFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdPipelineBarrier

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdPushConstants(3)

Name

vkCmdPushConstants - Update the values of push constants

C Specification

To update push constants, call:

```
void vkCmdPushConstants(
    VkCommandBuffer                             commandBuffer,
    VkPipelineLayout                            layout,
    VkShaderStageFlags                          stageFlags,
    uint32_t                                    offset,
    uint32_t                                    size,
    const void*                                 pValues);
```

Parameters

- `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 an array of `size` bytes containing the new push constant values.

Description

Valid Usage

- `stageFlags` **must** match exactly the shader stages used in `layout` for the range specified by `offset` and `size`
- `offset` **must** be a multiple of `4`
- `size` **must** be a multiple of `4`
- `offset` **must** be less than `VkPhysicalDeviceLimits`::`maxPushConstantsSize`
- `size` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxPushConstantsSize` minus `offset`

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `layout` **must** be a valid `VkPipelineLayout` handle
- `stageFlags` **must** be a valid combination of VkShaderStageFlagBits values
- `stageFlags` **must** not be `0`
- `pValues` **must** be a pointer to an array of `size` bytes
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- `size` **must** be greater than `0`
- 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	Pipeline Type
Primary Secondary	Both	Graphics compute	

See Also

VkCommandBuffer, VkPipelineLayout, VkShaderStageFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdPushConstants

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdResetEvent(3)

Name

vkCmdResetEvent - Reset an event object to non-signaled state

C Specification

To set the state of an event to unsignaled from a device, call:

```
void vkCmdResetEvent(
    VkCommandBuffer                             commandBuffer,
    VkEvent                                     event,
    VkPipelineStageFlags                        stageMask);
```

Parameters

- `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.

Description

When vkCmdResetEvent 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 every command previously submitted to the same queue, including those in the same command buffer and batch. The synchronization scope is limited to operations on the pipeline stages determined by the source stage mask specified by `stageMask`.

The second synchronization scope includes only the event unsignal operation.

If `event` is already in the unsignaled state when vkCmdResetEvent is executed on the device, then vkCmdResetEvent has no effect, no event unsignal operation occurs, and no execution dependency is generated.

Valid Usage

- `stageMask` **must** not include `VK_PIPELINE_STAGE_HOST_BIT`
- If the geometry shaders feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- If the tessellation shaders feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- When this command executes, `event` **must** not be waited on by a `vkCmdWaitEvents` command that is currently executing

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `event` **must** be a valid `VkEvent` handle
- `stageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- `stageMask` **must** not be `0`
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- This command **must** only be called outside of a render pass instance
- 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	Pipeline Type
Primary Secondary	Outside	Graphics compute	

See Also

VkCommandBuffer, VkEvent, VkPipelineStageFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdResetEvent

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdResetQueryPool(3)

Name

vkCmdResetQueryPool - Reset queries in a query pool

C Specification

To reset a range of queries in a query pool, call:

```
void vkCmdResetQueryPool(
    VkCommandBuffer                             commandBuffer,
    VkQueryPool                                 queryPool,
    uint32_t                                    firstQuery,
    uint32_t                                    queryCount);
```

Parameters

- `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.

Description

When executed on a queue, this command sets the status of query indices [`firstQuery`, `firstQuery` + `queryCount` - 1] to unavailable.

Valid Usage

- `firstQuery` **must** be less than the number of queries in `queryPool`
- The sum of `firstQuery` and `queryCount` **must** be less than or equal to the number of queries in `queryPool`

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `queryPool` **must** be a valid `VkQueryPool` handle
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- This command **must** only be called outside of a render pass instance
- 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	Pipeline Type
Primary Secondary	Outside	Graphics compute	

See Also

VkCommandBuffer, VkQueryPool

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdResetQueryPool

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdResolveImage(3)

Name

vkCmdResolveImage - Resolve regions of an image

C Specification

To resolve a multisample image to a non-multisample image, call:

```
void vkCmdResolveImage(
    VkCommandBuffer                             commandBuffer,
    VkImage                                     srcImage,
    VkImageLayout                               srcImageLayout,
    VkImage                                     dstImage,
    VkImageLayout                               dstImageLayout,
    uint32_t                                    regionCount,
    const VkImageResolve*                       pRegions);
```

Parameters

- `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.

Description

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 formats are floating-point or normalized types, the sample values for each pixel are resolved in an implementation-dependent manner. If the source formats are integer types, 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`.

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

- The source region specified by a given element of `pRegions` **must** be a region that is contained within `srcImage`
- The destination region specified by a given element of `pRegions` **must** be a region that is contained within `dstImage`
- The union of all source regions, and the union of all destination regions, specified by the elements of `pRegions`, **must** not overlap in memory
- If `srcImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- `srcImage` **must** have a sample count equal to any valid sample count value other than `VK_SAMPLE_COUNT_1_BIT`
- If `dstImage` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- `dstImage` **must** have a sample count equal to `VK_SAMPLE_COUNT_1_BIT`
- `srcImageLayout` **must** specify the layout of the image subresources of `srcImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- `srcImageLayout` **must** be `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- `dstImageLayout` **must** specify the layout of the image subresources of `dstImage` specified in `pRegions` at the time this command is executed on a `VkDevice`
- `dstImageLayout` **must** be `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` or `VK_IMAGE_LAYOUT_GENERAL`
- If `dstImage` was created with `tiling` equal to `VK_IMAGE_TILING_LINEAR`, `dstImage` **must** have been created with a `format` that supports being a color attachment, as specified by the `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` flag in `VkFormatProperties::linearTilingFeatures` returned by `vkGetPhysicalDeviceFormatProperties`
- If `dstImage` was created with `tiling` equal to `VK_IMAGE_TILING_OPTIMAL`, `dstImage` **must** have been created with a `format` that supports being a color attachment, as specified by the `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` flag in `VkFormatProperties::optimalTilingFeatures` returned by `vkGetPhysicalDeviceFormatProperties`
- `srcImage` and `dstImage` **must** have been created with the same image format

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `srcImage` **must** be a valid `VkImage` handle
- `srcImageLayout` **must** be a valid VkImageLayout value
- `dstImage` **must** be a valid `VkImage` handle
- `dstImageLayout` **must** be a valid VkImageLayout value
- `pRegions` **must** be a pointer to an array of `regionCount` valid `VkImageResolve` structures
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- This command **must** only be called outside of a render pass instance
- `regionCount` **must** be greater than `0`
- 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	Pipeline Type
Primary Secondary	Outside	Graphics	Transfer

See Also

VkCommandBuffer, VkImage, VkImageLayout, VkImageResolve

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdResolveImage

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the

Specification, not directly.

vkCmdSetBlendConstants(3)

Name

vkCmdSetBlendConstants - Set the values of blend constants

C Specification

Otherwise, to dynamically set and change the blend constant, call:

```
void vkCmdSetBlendConstants(
    VkCommandBuffer                             commandBuffer,
    const float                                 blendConstants[4]);
```

Parameters

- `commandBuffer` is the command buffer into which the command will be recorded.
- `blendConstants` is an array of four values specifying the R, G, B, and A components of the blend constant color used in blending, depending on the blend factor.

Description

Valid Usage

- The currently bound graphics pipeline **must** have been created with the `VK_DYNAMIC_STATE_BLEND_CONSTANTS` dynamic state enabled

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics 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	Pipeline Type
Primary Secondary	Both	Graphics	

See Also

VkCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdSetBlendConstants

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdSetDepthBias(3)

Name

vkCmdSetDepthBias - Set the depth bias dynamic state

C Specification

The depth values of all fragments generated by the rasterization of a polygon **can** be offset by a single value that is computed for that polygon. This behavior is controlled by the `depthBiasEnable`, `depthBiasConstantFactor`, `depthBiasClamp`, and `depthBiasSlopeFactor` members of VkPipelineRasterizationStateCreateInfo, or by the corresponding parameters to the `vkCmdSetDepthBias` command if depth bias state is dynamic.

```
void vkCmdSetDepthBias(
    VkCommandBuffer                             commandBuffer,
    float                                       depthBiasConstantFactor,
    float                                       depthBiasClamp,
    float                                       depthBiasSlopeFactor);
```

Parameters

- `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.

Description

If `depthBiasEnable` is `VK_FALSE`, no depth bias is applied and the fragment's depth values are unchanged.

`depthBiasSlopeFactor` scales the maximum depth slope of the polygon, and `depthBiasConstantFactor` scales an implementation-dependent constant that relates to the usable resolution of the depth buffer. The resulting values are summed to produce the depth bias 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.

The maximum depth slope m of a triangle is

$$m = \sqrt{\left(\frac{\partial z_f}{\partial x_f}\right)^2 + \left(\frac{\partial z_f}{\partial y_f}\right)^2}$$

where (x_f, y_f, z_f) is a point on the triangle. m **may** be approximated as

$$m = \max\left(\left|\frac{\partial z_f}{\partial x_f}\right|, \left|\frac{\partial z_f}{\partial y_f}\right|\right).$$

The minimum resolvable difference r is an implementation-dependent parameter that depends on the depth buffer representation. It is the smallest difference in framebuffer coordinate z values that is guaranteed to remain distinct throughout polygon rasterization and in the depth buffer. 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 buffer representations, r is constant throughout the range of the entire depth buffer. For floating-point depth buffers, 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 buffer is present, r is undefined.

The bias value o for a polygon is

$$o = \begin{cases} m \times depthBiasSlopeFactor + r \times depthBiasConstantFactor & depthBiasClamp = 0 \text{ or } NaN \\ \min(m \times depthBiasSlopeFactor + r \times depthBiasConstantFactor,\ depthBiasClamp) & depthBiasClamp > 0 \\ \max(m \times depthBiasSlopeFactor + r \times depthBiasConstantFactor,\ depthBiasClamp) & depthBiasClamp < 0 \end{cases}$$

m is computed as described above. If the depth buffer 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.

For fixed-point depth buffers, fragment depth values are always limited to the range [0,1] by clamping after depth bias addition is performed. Fragment depth values are clamped even when the depth buffer uses a floating-point representation.

Valid Usage

- The currently bound graphics pipeline **must** have been created with the `VK_DYNAMIC_STATE_DEPTH_BIAS` dynamic state enabled
- If the depth bias clamping feature is not enabled, `depthBiasClamp` **must** be `0`.0

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics 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	Pipeline Type
Primary Secondary	Both	Graphics	

See Also

VkCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdSetDepthBias

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdSetDepthBounds(3)

Name

vkCmdSetDepthBounds - Set the depth bounds test values for a command buffer

C Specification

The depth bounds test conditionally disables coverage of a sample based on the outcome of a comparison between the value z_a in the depth attachment at location (x_f,y_f) (for the appropriate sample) and a range of values. The test is enabled or disabled by the `depthBoundsTestEnable` member of VkPipelineDepthStencilStateCreateInfo: If the pipeline state object is created without the `VK_DYNAMIC_STATE_DEPTH_BOUNDS` dynamic state enabled then the range of values used in the depth bounds test are defined by the `minDepthBounds` and `maxDepthBounds` members of the VkPipelineDepthStencilStateCreateInfo structure. Otherwise, to dynamically set the depth bounds range values call:

```
void vkCmdSetDepthBounds(
    VkCommandBuffer                             commandBuffer,
    float                                       minDepthBounds,
    float                                       maxDepthBounds);
```

Parameters

- `commandBuffer` is the command buffer into which the command will be recorded.
- `minDepthBounds` is the lower bound of the range of depth values used in the depth bounds test.
- `maxDepthBounds` is the upper bound of the range.

Description

Valid Usage

- The currently bound graphics pipeline **must** have been created with the `VK_DYNAMIC_STATE_DEPTH_BOUNDS` dynamic state enabled
- `minDepthBounds` **must** be between `0.0` and `1.0`, inclusive
- `maxDepthBounds` **must** be between `0.0` and `1.0`, inclusive

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics 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	Pipeline Type
Primary Secondary	Both	Graphics	

See Also

VkCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdSetDepthBounds

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdSetEvent(3)

Name

vkCmdSetEvent - Set an event object to signaled state

C Specification

To set the state of an event to signaled from a device, call:

```
void vkCmdSetEvent(
    VkCommandBuffer                             commandBuffer,
    VkEvent                                     event,
    VkPipelineStageFlags                        stageMask);
```

Parameters

- `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 when the `event` is signaled.

Description

When vkCmdSetEvent is submitted to a queue, it defines an execution dependency on commands that were submitted before it, and defines an event signal operation which sets the event to the signaled state.

The first synchronization scope includes every command previously submitted to the same queue, including those in the same command buffer and batch. The synchronization scope is limited to operations on the pipeline stages determined by the source stage mask specified by `stageMask`.

The second synchronization scope includes only the event signal operation.

If `event` is already in the signaled state when vkCmdSetEvent is executed on the device, then vkCmdSetEvent has no effect, no event signal operation occurs, and no execution dependency is generated.

Valid Usage

- `stageMask` **must** not include `VK_PIPELINE_STAGE_HOST_BIT`
- If the geometry shaders feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- If the tessellation shaders feature is not enabled, `stageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `event` **must** be a valid `VkEvent` handle
- `stageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- `stageMask` **must** not be `0`
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- This command **must** only be called outside of a render pass instance
- 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	Pipeline Type
Primary Secondary	Outside	Graphics compute	

See Also

VkCommandBuffer, VkEvent, VkPipelineStageFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdSetEvent

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdSetLineWidth(3)

Name

vkCmdSetLineWidth - Set the dynamic line width state

C Specification

The line width is specified by the VkPipelineRasterizationStateCreateInfo::`lineWidth` property of the currently active pipeline, if the pipeline was not created with `VK_DYNAMIC_STATE_LINE_WIDTH` enabled.

Otherwise, the line width is set by calling `vkCmdSetLineWidth`:

```
void vkCmdSetLineWidth(
    VkCommandBuffer                             commandBuffer,
    float                                       lineWidth);
```

Parameters

- `commandBuffer` is the command buffer into which the command will be recorded.
- `lineWidth` is the width of rasterized line segments.

Description

Valid Usage

- The currently bound graphics pipeline **must** have been created with the `VK_DYNAMIC_STATE_LINE_WIDTH` dynamic state enabled
- If the wide lines feature is not enabled, `lineWidth` **must** be `1.0`

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics 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	Pipeline Type
Primary Secondary	Both	Graphics	

See Also

VkCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdSetLineWidth

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdSetScissor(3)

Name

vkCmdSetScissor - Set the dynamic scissor rectangles on a command buffer

C Specification

The scissor test determines if a fragment's framebuffer coordinates (x_f,y_f) lie within the scissor rectangle corresponding to the viewport index (see Controlling the Viewport) used by the primitive that generated the fragment. If the pipeline state object is created without `VK_DYNAMIC_STATE_SCISSOR` enabled then the scissor rectangles are set by the VkPipelineViewportStateCreateInfo state of the pipeline state object. Otherwise, to dynamically set the scissor rectangles call:

```
void vkCmdSetScissor(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstScissor,
    uint32_t                                    scissorCount,
    const VkRect2D*                             pScissors);
```

Parameters

- `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.

Description

The scissor rectangles taken from element i of `pScissors` replace the current state for the scissor index `firstScissor` + i, for i in [0, `scissorCount`).

Each scissor rectangle is described by a VkRect2D structure, with the `offset.x` and `offset.y` values determining the upper left corner of the scissor rectangle, and the `extent.width` and `extent.height` values determining the size in pixels.

Valid Usage

- The currently bound graphics pipeline **must** have been created with the `VK_DYNAMIC_STATE_SCISSOR` dynamic state enabled
- `firstScissor` **must** be less than `VkPhysicalDeviceLimits`::`maxViewports`
- The sum of `firstScissor` and `scissorCount` **must** be between `1` and `VkPhysicalDeviceLimits`::`maxViewports`, inclusive
- If the multiple viewports feature is not enabled, `firstScissor` **must** be `0`
- If the multiple viewports feature is not enabled, `scissorCount` **must** be `1`
- The `x` and `y` members of `offset` **must** be greater than or equal to `0`
- Evaluation of (`offset.x` + `extent.width`) **must** not cause a signed integer addition overflow
- Evaluation of (`offset.y` + `extent.height`) **must** not cause a signed integer addition overflow

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `pScissors` **must** be a pointer to an array of `scissorCount` `VkRect2D` structures
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- `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	Pipeline Type
Primary Secondary	Both	Graphics	

See Also

VkCommandBuffer, VkRect2D

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdSetScissor

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdSetStencilCompareMask(3)

Name

vkCmdSetStencilCompareMask - Set the stencil compare mask dynamic state

C Specification

If the pipeline state object is created with the `VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK` dynamic state enabled, then to dynamically set the stencil compare mask call:

```
void vkCmdSetStencilCompareMask(
    VkCommandBuffer                             commandBuffer,
    VkStencilFaceFlags                          faceMask,
    uint32_t                                    compareMask);
```

Parameters

- `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.

Description

Valid Usage

- The currently bound graphics pipeline **must** have been created with the `VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK` dynamic state enabled

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `faceMask` **must** be a valid combination of VkStencilFaceFlagBits values
- `faceMask` **must** not be `0`
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics 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	Pipeline Type
Primary Secondary	Both	Graphics	

See Also

VkCommandBuffer, VkStencilFaceFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdSetStencilCompareMask

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdSetStencilReference(3)

Name

vkCmdSetStencilReference - Set the stencil reference dynamic state

C Specification

If the pipeline state object is created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled, then to dynamically set the stencil reference value call:

```
void vkCmdSetStencilReference(
    VkCommandBuffer                             commandBuffer,
    VkStencilFaceFlags                          faceMask,
    uint32_t                                    reference);
```

Parameters

- 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.

Description

Valid Usage

- The currently bound graphics pipeline **must** have been created with the VK_DYNAMIC_STATE_STENCIL_REFERENCE dynamic state enabled

Valid Usage (Implicit)

- commandBuffer **must** be a valid VkCommandBuffer handle
- faceMask **must** be a valid combination of VkStencilFaceFlagBits values
- faceMask **must** not be 0
- commandBuffer **must** be in the recording state
- The VkCommandPool that commandBuffer was allocated from **must** support graphics 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	Pipeline Type
Primary Secondary	Both	Graphics	

See Also

VkCommandBuffer, VkStencilFaceFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdSetStencilReference

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdSetStencilWriteMask(3)

Name

vkCmdSetStencilWriteMask - Set the stencil write mask dynamic state

C Specification

If the pipeline state object is created with the `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` dynamic state enabled, then to dynamically set the stencil write mask call:

```
void vkCmdSetStencilWriteMask(
    VkCommandBuffer                             commandBuffer,
    VkStencilFaceFlags                          faceMask,
    uint32_t                                    writeMask);
```

Parameters

- `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.

Description

Valid Usage

- The currently bound graphics pipeline **must** have been created with the `VK_DYNAMIC_STATE_STENCIL_WRITE_MASK` dynamic state enabled

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `faceMask` **must** be a valid combination of VkStencilFaceFlagBits values
- `faceMask` **must** not be `0`
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics 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	Pipeline Type
Primary Secondary	Both	Graphics	

See Also

VkCommandBuffer, VkStencilFaceFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdSetStencilWriteMask

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdSetViewport(3)

Name

vkCmdSetViewport - Set the viewport on a command buffer

C Specification

If the bound pipeline state object was not created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled, viewport transformation parameters are specified using the `pViewports` member of `VkPipelineViewportStateCreateInfo` in the pipeline state object. If the pipeline state object was created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled, the viewport transformation parameters are dynamically set and changed with the command:

```
void vkCmdSetViewport(
    VkCommandBuffer                             commandBuffer,
    uint32_t                                    firstViewport,
    uint32_t                                    viewportCount,
    const VkViewport*                           pViewports);
```

Parameters

- `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.

Description

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

- The currently bound graphics pipeline **must** have been created with the `VK_DYNAMIC_STATE_VIEWPORT` dynamic state enabled
- `firstViewport` **must** be less than `VkPhysicalDeviceLimits`::`maxViewports`
- The sum of `firstViewport` and `viewportCount` **must** be between `1` and `VkPhysicalDeviceLimits`::`maxViewports`, inclusive
- If the multiple viewports feature is not enabled, `firstViewport` **must** be `0`
- If the multiple viewports feature is not enabled, `viewportCount` **must** be `1`
- `pViewports` **must** be a pointer to an array of `viewportCount` valid `VkViewport` structures

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics operations
- `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	Pipeline Type
Primary Secondary	Both	Graphics	

See Also

VkCommandBuffer, VkViewport

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdSetViewport

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdUpdateBuffer(3)

Name

vkCmdUpdateBuffer - Update a buffer's contents from host memory

C Specification

To update buffer data inline in a command buffer, call:

```
void vkCmdUpdateBuffer(
    VkCommandBuffer                             commandBuffer,
    VkBuffer                                    dstBuffer,
    VkDeviceSize                                dstOffset,
    VkDeviceSize                                dataSize,
    const void*                                 pData);
```

Parameters

- `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.

Description

`dataSize` **must** be less than or equal to 65536 bytes. For larger updates, applications **can** use buffer to buffer copies.

The source data is copied from the user pointer to the command buffer when the command is called.

`vkCmdUpdateBuffer` is only allowed outside of a render pass. This command is treated as "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

- `dstOffset` **must** be less than the size of `dstBuffer`
- `dataSize` **must** be less than or equal to the size of `dstBuffer` minus `dstOffset`
- `dstBuffer` **must** have been created with `VK_BUFFER_USAGE_TRANSFER_DST_BIT` usage flag
- If `dstBuffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object
- `dstOffset` **must** be a multiple of `4`
- `dataSize` **must** be less than or equal to `65536`
- `dataSize` **must** be a multiple of `4`

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `dstBuffer` **must** be a valid `VkBuffer` handle
- `pData` **must** be a pointer to an array of `dataSize` bytes
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support transfer, graphics, or compute operations
- This command **must** only be called outside of a render pass instance
- `dataSize` **must** be greater than `0`
- 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	Pipeline Type
Primary Secondary	Outside	Transfer graphics compute	Transfer

See Also

VkBuffer, VkCommandBuffer, `VkDeviceSize`

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdUpdateBuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdWaitEvents(3)

Name

vkCmdWaitEvents - Wait for one or more events and insert a set of memory

C Specification

To wait for one or more events to enter the signaled state on a device, call:

```
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);
```

Parameters

- `commandBuffer` is the command buffer into which the command is recorded.
- `eventCount` is the length of the `pEvents` array.
- `pEvents` is 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.

Description

When `vkCmdWaitEvents` is submitted to a queue, it defines a memory dependency between prior event signal operations, and subsequent commands.

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 were previously submitted to the same queue 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 commands subsequently submitted to the same queue, including those in the same command buffer and batch. 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 access 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 access 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.

> *Note*
>
> vkCmdWaitEvents is used with vkCmdSetEvent 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 a vkCmdResetEvent command and a vkCmdWaitEvents command submitted after it, so some other execution dependency **must** be included between these commands (e.g. a semaphore).

Valid Usage

- `srcStageMask` **must** be the bitwise OR of the `stageMask` parameter used in previous calls to `vkCmdSetEvent` with any of the members of `pEvents` and `VK_PIPELINE_STAGE_HOST_BIT` if any of the members of `pEvents` was set using `vkSetEvent`
- If the geometry shaders feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- If the geometry shaders feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- If the tessellation shaders feature is not enabled, `srcStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- If the tessellation shaders feature is not enabled, `dstStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- If `pEvents` includes one or more events that will be signaled by `vkSetEvent` after `commandBuffer` has been submitted to a queue, then `vkCmdWaitEvents` **must** not be called inside a render pass instance
- Any pipeline stage included in `srcStageMask` or `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.
- Any given element of `pMemoryBarriers`, `pBufferMemoryBarriers` or `pImageMemoryBarriers` **must** not have any access flag included in its `srcAccessMask` member if that bit is not supported by any of the pipeline stages in `srcStageMask`, as specified in the table of supported access types.
- Any given element of `pMemoryBarriers`, `pBufferMemoryBarriers` or `pImageMemoryBarriers` **must** not have any access flag included in its `dstAccessMask` member if that bit is not supported by any of the pipeline stages in `dstStageMask`, as specified in the table of supported access types.

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `pEvents` **must** be a pointer to an array of `eventCount` valid `VkEvent` handles
- `srcStageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- `srcStageMask` **must** not be `0`
- `dstStageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- `dstStageMask` **must** not be `0`
- If `memoryBarrierCount` is not `0`, `pMemoryBarriers` **must** be a pointer to an array of `memoryBarrierCount` valid `VkMemoryBarrier` structures
- If `bufferMemoryBarrierCount` is not `0`, `pBufferMemoryBarriers` **must** be a pointer to an array of `bufferMemoryBarrierCount` valid `VkBufferMemoryBarrier` structures
- If `imageMemoryBarrierCount` is not `0`, `pImageMemoryBarriers` **must** be a pointer to an array of `imageMemoryBarrierCount` valid `VkImageMemoryBarrier` structures
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- `eventCount` **must** be greater than `0`
- 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	Pipeline Type
Primary Secondary	Both	Graphics compute	

See Also

VkBufferMemoryBarrier, VkCommandBuffer, VkEvent, VkImageMemoryBarrier, VkMemoryBarrier, VkPipelineStageFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdWaitEvents

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCmdWriteTimestamp(3)

Name

vkCmdWriteTimestamp - Write a device timestamp into a query object

C Specification

To request a timestamp, call:

```
void vkCmdWriteTimestamp(
    VkCommandBuffer                             commandBuffer,
    VkPipelineStageFlagBits                     pipelineStage,
    VkQueryPool                                 queryPool,
    uint32_t                                    query);
```

Parameters

- `commandBuffer` is the command buffer into which the command will be recorded.
- `pipelineStage` is one of the VkPipelineStageFlagBits, 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.

Description

`vkCmdWriteTimestamp` latches the value of the timer when all previous commands have completed executing as far as the specified pipeline stage, and writes the timestamp value to memory. When the timestamp value is written, the availability status of the query is set to available.

> *Note*
>
> If an implementation is unable to detect completion and latch the timer at any specific stage of the pipeline, it **may** instead do so at any logically later stage.

vkCmdCopyQueryPoolResults **can** then be called to copy the timestamp value from the query pool into buffer memory, with ordering and synchronization behavior equivalent to how other queries operate. Timestamp values **can** also be retrieved from the query pool using vkGetQueryPoolResults. As with other queries, the query **must** be reset using vkCmdResetQueryPool before requesting the timestamp value be written to it.

While `vkCmdWriteTimestamp` **can** be called inside or outside of a render pass instance, vkCmdCopyQueryPoolResults **must** only be called outside of a render pass instance.

Valid Usage

- `queryPool` **must** have been created with a `queryType` of `VK_QUERY_TYPE_TIMESTAMP`
- The query identified by `queryPool` and `query` **must** be *unavailable*
- The command pool's queue family **must** support a non-zero `timestampValidBits`

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `pipelineStage` **must** be a valid VkPipelineStageFlagBits value
- `queryPool` **must** be a valid `VkQueryPool` handle
- `commandBuffer` **must** be in the recording state
- The `VkCommandPool` that `commandBuffer` was allocated from **must** support graphics, or compute operations
- 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	Pipeline Type
Primary Secondary	Both	Graphics compute	Transfer

See Also

VkCommandBuffer, VkPipelineStageFlagBits, VkQueryPool

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCmdWriteTimestamp

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateBuffer(3)

Name

vkCreateBuffer - Create a new buffer object

C Specification

To create buffers, call:

```
VkResult vkCreateBuffer(
    VkDevice                                    device,
    const VkBufferCreateInfo*                   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkBuffer*                                   pBuffer);
```

Parameters

- `device` is the logical device that creates the buffer object.
- `pCreateInfo` is a pointer to an instance of the `VkBufferCreateInfo` structure containing parameters affecting creation of the buffer.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pBuffer` points to a `VkBuffer` handle in which the resulting buffer object is returned.

Description

Valid Usage

- If the `flags` member of `pCreateInfo` includes `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`, creating this `VkBuffer` **must** not cause the total required sparse memory for all currently valid sparse resources on the device to exceed `VkPhysicalDeviceLimits` ::`sparseAddressSpaceSize`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkBufferCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pBuffer` **must** be a pointer to a `VkBuffer` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkBuffer, VkBufferCreateInfo, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateBuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateBufferView(3)

Name

vkCreateBufferView - Create a new buffer view object

C Specification

To create a buffer view, call:

```
VkResult vkCreateBufferView(
    VkDevice                                    device,
    const VkBufferViewCreateInfo*               pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkBufferView*                               pView);
```

Parameters

- `device` is the logical device that creates the buffer view.
- `pCreateInfo` is a pointer to an instance of the `VkBufferViewCreateInfo` structure containing parameters to be used to create the buffer.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pView` points to a `VkBufferView` handle in which the resulting buffer view object is returned.

Description

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkBufferViewCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pView` **must** be a pointer to a `VkBufferView` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkBufferView, VkBufferViewCreateInfo, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateBufferView

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateCommandPool(3)

Name

vkCreateCommandPool - Create a new command pool object

C Specification

To create a command pool, call:

```
VkResult vkCreateCommandPool(
    VkDevice                                    device,
    const VkCommandPoolCreateInfo*              pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkCommandPool*                              pCommandPool);
```

Parameters

- `device` is the logical device that creates the command pool.
- `pCreateInfo` contains information used to create the command pool.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pCommandPool` points to a `VkCommandPool` handle in which the created pool is returned.

Description

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkCommandPoolCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pCommandPool` **must** be a pointer to a `VkCommandPool` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkCommandPool, VkCommandPoolCreateInfo, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateCommandPool

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateComputePipelines(3)

Name

vkCreateComputePipelines - Creates a new compute pipeline object

C Specification

To create compute pipelines, call:

```
VkResult vkCreateComputePipelines(
    VkDevice                                    device,
    VkPipelineCache                             pipelineCache,
    uint32_t                                    createInfoCount,
    const VkComputePipelineCreateInfo*          pCreateInfos,
    const VkAllocationCallbacks*                pAllocator,
    VkPipeline*                                 pPipelines);
```

Parameters

- `device` is the logical device that creates the compute pipelines.
- `pipelineCache` is either VK_NULL_HANDLE, indicating that pipeline caching is disabled; or the handle of a valid pipeline cache object, in which case use of that cache is enabled for the duration of the command.
- `createInfoCount` is the length of the `pCreateInfos` and `pPipelines` arrays.
- `pCreateInfos` is an array of `VkComputePipelineCreateInfo` structures.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pPipelines` is a pointer to an array in which the resulting compute pipeline objects are returned.

Description

Valid Usage

- If the `flags` member of any given element of `pCreateInfos` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, and the `basePipelineIndex` member of that same element is not `-1`, `basePipelineIndex` **must** be less than the index into `pCreateInfos` that corresponds to that element
- If the `flags` member of any given element of `pCreateInfos` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, the base pipeline **must** have been created with the `VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT` flag set

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `pipelineCache` is not VK_NULL_HANDLE, `pipelineCache` **must** be a valid `VkPipelineCache` handle
- `pCreateInfos` **must** be a pointer to an array of `createInfoCount` valid `VkComputePipelineCreateInfo` structures
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pPipelines` **must** be a pointer to an array of `createInfoCount` `VkPipeline` handles
- `createInfoCount` **must** be greater than `0`
- If `pipelineCache` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkComputePipelineCreateInfo, VkDevice, VkPipeline, VkPipelineCache

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateComputePipelines

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateDescriptorPool(3)

Name

vkCreateDescriptorPool - Creates a descriptor pool object

C Specification

To create a descriptor pool object, call:

```
VkResult vkCreateDescriptorPool(
    VkDevice                                    device,
    const VkDescriptorPoolCreateInfo*           pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDescriptorPool*                           pDescriptorPool);
```

Parameters

- `device` is the logical device that creates the descriptor pool.
- `pCreateInfo` is a pointer to an instance of the VkDescriptorPoolCreateInfo structure specifying the state of the descriptor pool object.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pDescriptorPool` points to a `VkDescriptorPool` handle in which the resulting descriptor pool object is returned.

Description

`pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

The created descriptor pool is returned in `pDescriptorPool`.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkDescriptorPoolCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pDescriptorPool` **must** be a pointer to a `VkDescriptorPool` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkDescriptorPool, VkDescriptorPoolCreateInfo, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateDescriptorPool

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateDescriptorSetLayout(3)

Name

vkCreateDescriptorSetLayout - Create a new descriptor set layout

C Specification

To create descriptor set layout objects, call:

```
VkResult vkCreateDescriptorSetLayout(
    VkDevice                                    device,
    const VkDescriptorSetLayoutCreateInfo*      pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDescriptorSetLayout*                      pSetLayout);
```

Parameters

- `device` is the logical device that creates the descriptor set layout.
- `pCreateInfo` is a pointer to an instance of the VkDescriptorSetLayoutCreateInfo structure specifying the state of the descriptor set layout object.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pSetLayout` points to a `VkDescriptorSetLayout` handle in which the resulting descriptor set layout object is returned.

Description

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkDescriptorSetLayoutCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pSetLayout` **must** be a pointer to a `VkDescriptorSetLayout` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkDescriptorSetLayout, VkDescriptorSetLayoutCreateInfo, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateDescriptorSetLayout

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateDevice(3)

Name

vkCreateDevice - Create a new device instance

C Specification

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

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

Parameters

- `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` points to a handle in which the created `VkDevice` is returned.

Description

`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.

After verifying and enabling the extensions the `VkDevice` 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.

Multiple logical devices **can** be created from the same physical device. Logical device creation **may** fail due to lack of device-specific resources (in addition to the other errors). If that occurs, `vkCreateDevice` will return `VK_ERROR_TOO_MANY_OBJECTS`.

Valid Usage

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

Valid Usage (Implicit)

- `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkDeviceCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pDevice` **must** be a pointer to a `VkDevice` handle

Return Codes

Success
- `VK_SUCCESS`

Failure
- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_INITIALIZATION_FAILED`
- `VK_ERROR_EXTENSION_NOT_PRESENT`
- `VK_ERROR_FEATURE_NOT_PRESENT`
- `VK_ERROR_TOO_MANY_OBJECTS`
- `VK_ERROR_DEVICE_LOST`

See Also

VkAllocationCallbacks, VkDevice, VkDeviceCreateInfo, VkPhysicalDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateDevice

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateEvent(3)

Name

vkCreateEvent - Create a new event object

C Specification

To create an event, call:

```
VkResult vkCreateEvent(
    VkDevice                                    device,
    const VkEventCreateInfo*                    pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkEvent*                                    pEvent);
```

Parameters

- `device` is the logical device that creates the event.
- `pCreateInfo` is a pointer to an instance of the `VkEventCreateInfo` structure which contains information about how the event is to be created.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pEvent` points to a handle in which the resulting event object is returned.

Description

When created, the event object is in the unsignaled state.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkEventCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pEvent` **must** be a pointer to a `VkEvent` handle

Return Codes

Success

- VK_SUCCESS

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

See Also

VkAllocationCallbacks, VkDevice, VkEvent, VkEventCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateEvent

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateFence(3)

Name

vkCreateFence - Create a new fence object

C Specification

To create a fence, call:

```
VkResult vkCreateFence(
    VkDevice                                    device,
    const VkFenceCreateInfo*                    pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkFence*                                    pFence);
```

Parameters

- `device` is the logical device that creates the fence.
- `pCreateInfo` is a pointer to an instance of the `VkFenceCreateInfo` structure which contains information about how the fence is to be created.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pFence` points to a handle in which the resulting fence object is returned.

Description

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkFenceCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pFence` **must** be a pointer to a `VkFence` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkDevice, VkFence, VkFenceCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateFence

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateFramebuffer(3)

Name

vkCreateFramebuffer - Create a new framebuffer object

C Specification

To create a framebuffer, call:

```
VkResult vkCreateFramebuffer(
    VkDevice                                    device,
    const VkFramebufferCreateInfo*              pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkFramebuffer*                              pFramebuffer);
```

Parameters

- `device` is the logical device that creates the framebuffer.
- `pCreateInfo` points to a VkFramebufferCreateInfo structure which describes additional information about framebuffer creation.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pFramebuffer` points to a `VkFramebuffer` handle in which the resulting framebuffer object is returned.

Description

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkFramebufferCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pFramebuffer` **must** be a pointer to a `VkFramebuffer` handle

Return Codes

Success

- VK_SUCCESS

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

See Also

VkAllocationCallbacks, VkDevice, VkFramebuffer, VkFramebufferCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateFramebuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateGraphicsPipelines(3)

Name

vkCreateGraphicsPipelines - Create graphics pipelines

C Specification

To create graphics pipelines, call:

```
VkResult vkCreateGraphicsPipelines(
    VkDevice                                    device,
    VkPipelineCache                             pipelineCache,
    uint32_t                                    createInfoCount,
    const VkGraphicsPipelineCreateInfo*         pCreateInfos,
    const VkAllocationCallbacks*                pAllocator,
    VkPipeline*                                 pPipelines);
```

Parameters

- `device` is the logical device that creates the graphics pipelines.
- `pipelineCache` is either VK_NULL_HANDLE, indicating that pipeline caching is disabled; or the handle of a valid pipeline cache object, in which case use of that cache is enabled for the duration of the command.
- `createInfoCount` is the length of the `pCreateInfos` and `pPipelines` arrays.
- `pCreateInfos` is an array of `VkGraphicsPipelineCreateInfo` structures.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pPipelines` is a pointer to an array in which the resulting graphics pipeline objects are returned.

Description

The VkGraphicsPipelineCreateInfo structure includes an array of shader create info structures containing all the desired active shader stages, as well as creation info to define all relevant fixed-function stages, and a pipeline layout.

Valid Usage

- If the flags member of any given element of pCreateInfos contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and the basePipelineIndex member of that same element is not -1, basePipelineIndex **must** be less than the index into pCreateInfos that corresponds to that element
- If the flags member of any given element of pCreateInfos contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, the base pipeline **must** have been created with the VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT flag set

Valid Usage (Implicit)

- device **must** be a valid VkDevice handle
- If pipelineCache is not VK_NULL_HANDLE, pipelineCache **must** be a valid VkPipelineCache handle
- pCreateInfos **must** be a pointer to an array of createInfoCount valid VkGraphicsPipelineCreateInfo structures
- If pAllocator is not NULL, pAllocator **must** be a pointer to a valid VkAllocationCallbacks structure
- pPipelines **must** be a pointer to an array of createInfoCount VkPipeline handles
- createInfoCount **must** be greater than 0
- If pipelineCache is a valid handle, it **must** have been created, allocated, or retrieved from device

Return Codes

Success

- VK_SUCCESS

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

See Also

VkAllocationCallbacks, VkDevice, VkGraphicsPipelineCreateInfo, VkPipeline, VkPipelineCache

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateGraphicsPipelines

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateImage(3)

Name

vkCreateImage - Create a new image object

C Specification

To create images, call:

```
VkResult vkCreateImage(
    VkDevice                                    device,
    const VkImageCreateInfo*                    pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkImage*                                    pImage);
```

Parameters

- `device` is the logical device that creates the image.
- `pCreateInfo` is a pointer to an instance of the `VkImageCreateInfo` structure containing parameters to be used to create the image.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pImage` points to a `VkImage` handle in which the resulting image object is returned.

Description

Valid Usage

- If the `flags` member of `pCreateInfo` includes `VK_IMAGE_CREATE_SPARSE_BINDING_BIT`, creating this `VkImage` **must** not cause the total required sparse memory for all currently valid sparse resources on the device to exceed `VkPhysicalDeviceLimits`::`sparseAddressSpaceSize`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkImageCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pImage` **must** be a pointer to a `VkImage` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkDevice, VkImage, VkImageCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateImage

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateImageView(3)

Name

vkCreateImageView - Create an image view from an existing image

C Specification

To create an image view, call:

```
VkResult vkCreateImageView(
    VkDevice                                    device,
    const VkImageViewCreateInfo*                pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkImageView*                                pView);
```

Parameters

- `device` is the logical device that creates the image view.
- `pCreateInfo` is a pointer to an instance of the `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` points to a `VkImageView` handle in which the resulting image view object is returned.

Description

Some of the image creation parameters are inherited by the view. The remaining parameters are contained in the `pCreateInfo`.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkImageViewCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pView` **must** be a pointer to a `VkImageView` handle

Return Codes

Success

- VK_SUCCESS

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

See Also

VkAllocationCallbacks, VkDevice, VkImageView, VkImageViewCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateImageView

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateInstance(3)

Name

vkCreateInstance - Create a new Vulkan instance

C Specification

To create an instance object, call:

```
VkResult vkCreateInstance(
    const VkInstanceCreateInfo*                 pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkInstance*                                 pInstance);
```

Parameters

- `pCreateInfo` points to an instance of VkInstanceCreateInfo 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.

Description

`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

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

Valid Usage (Implicit)

- `pCreateInfo` **must** be a pointer to a valid `VkInstanceCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pInstance` **must** be a pointer to a `VkInstance` handle

Return Codes

Success

- VK_SUCCESS

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_INITIALIZATION_FAILED
- VK_ERROR_LAYER_NOT_PRESENT
- VK_ERROR_EXTENSION_NOT_PRESENT
- VK_ERROR_INCOMPATIBLE_DRIVER

See Also

VkAllocationCallbacks, VkInstance, VkInstanceCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateInstance

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreatePipelineCache(3)

Name

vkCreatePipelineCache - Creates a new pipeline cache

C Specification

To create pipeline cache objects, call:

```
VkResult vkCreatePipelineCache(
    VkDevice                                    device,
    const VkPipelineCacheCreateInfo*            pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkPipelineCache*                            pPipelineCache);
```

Parameters

- `device` is the logical device that creates the pipeline cache object.
- `pCreateInfo` is a pointer to a `VkPipelineCacheCreateInfo` structure that contains the 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.

Description

> *Note*
>
> Applications **can** track and manage the total host memory size of a pipeline cache object using the `pAllocator`. Applications **can** limit the amount of data retrieved from a pipeline cache object in `vkGetPipelineCacheData`. Implementations **should** not internally limit the total number of entries added to a pipeline cache object or the total host memory consumed.

Once created, a pipeline cache **can** be passed to the `vkCreateGraphicsPipelines` and `vkCreateComputePipelines` commands. If the pipeline cache passed into these commands is not VK_NULL_HANDLE, the implementation will query it for possible reuse opportunities and update it with new content. 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.

> *Note*
>
> Implementations **should** make every effort to limit any critical sections to the actual accesses to the cache, which is expected to be significantly shorter than the duration of the `vkCreateGraphicsPipelines` and `vkCreateComputePipelines` commands.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkPipelineCacheCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pPipelineCache` **must** be a pointer to a `VkPipelineCache` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkDevice, VkPipelineCache, VkPipelineCacheCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreatePipelineCache

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreatePipelineLayout(3)

Name

vkCreatePipelineLayout - Creates a new pipeline layout object

C Specification

To create a pipeline layout, call:

```
VkResult vkCreatePipelineLayout(
    VkDevice                                    device,
    const VkPipelineLayoutCreateInfo*           pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkPipelineLayout*                           pPipelineLayout);
```

Parameters

- `device` is the logical device that creates the pipeline layout.
- `pCreateInfo` is a pointer to an instance of the VkPipelineLayoutCreateInfo structure specifying the state of the pipeline layout object.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pPipelineLayout` points to a `VkPipelineLayout` handle in which the resulting pipeline layout object is returned.

Description

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkPipelineLayoutCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pPipelineLayout` **must** be a pointer to a `VkPipelineLayout` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkDevice, VkPipelineLayout, VkPipelineLayoutCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreatePipelineLayout

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateQueryPool(3)

Name

vkCreateQueryPool - Create a new query pool object

C Specification

To create a query pool, call:

```
VkResult vkCreateQueryPool(
    VkDevice                                    device,
    const VkQueryPoolCreateInfo*                pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkQueryPool*                                pQueryPool);
```

Parameters

- `device` is the logical device that creates the query pool.
- `pCreateInfo` is a pointer to an instance of the `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.

Description

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkQueryPoolCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pQueryPool` **must** be a pointer to a `VkQueryPool` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkDevice, VkQueryPool, VkQueryPoolCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateQueryPool

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateRenderPass(3)

Name

vkCreateRenderPass - Create a new render pass object

C Specification

To create a render pass, call:

```
VkResult vkCreateRenderPass(
    VkDevice                                    device,
    const VkRenderPassCreateInfo*               pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkRenderPass*                               pRenderPass);
```

Parameters

- `device` is the logical device that creates the render pass.
- `pCreateInfo` is a pointer to an instance of the VkRenderPassCreateInfo structure that describes the parameters of the render pass.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pRenderPass` points to a `VkRenderPass` handle in which the resulting render pass object is returned.

Description

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkRenderPassCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pRenderPass` **must** be a pointer to a `VkRenderPass` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkDevice, VkRenderPass, VkRenderPassCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateRenderPass

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateSampler(3)

Name

vkCreateSampler - Create a new sampler object

C Specification

To create a sampler object, call:

```
VkResult vkCreateSampler(
    VkDevice                                    device,
    const VkSamplerCreateInfo*                  pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSampler*                                  pSampler);
```

Parameters

- `device` is the logical device that creates the sampler.
- `pCreateInfo` is a pointer to an instance of the VkSamplerCreateInfo structure specifying the state of the sampler object.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pSampler` points to a VkSampler handle in which the resulting sampler object is returned.

Description

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkSamplerCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pSampler` **must** be a pointer to a `VkSampler` handle

Return Codes

Success

- VK_SUCCESS

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_TOO_MANY_OBJECTS

See Also

VkAllocationCallbacks, VkDevice, VkSampler, VkSamplerCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateSampler

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateSemaphore(3)

Name

vkCreateSemaphore - Create a new queue semaphore object

C Specification

To create a semaphore, call:

```
VkResult vkCreateSemaphore(
    VkDevice                                    device,
    const VkSemaphoreCreateInfo*                pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkSemaphore*                                pSemaphore);
```

Parameters

- `device` is the logical device that creates the semaphore.
- `pCreateInfo` is a pointer to an instance of the `VkSemaphoreCreateInfo` structure which contains information about how the semaphore is to be created.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pSemaphore` points to a handle in which the resulting semaphore object is returned.

Description

When created, the semaphore is in the unsignaled state.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkSemaphoreCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pSemaphore` **must** be a pointer to a `VkSemaphore` handle

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkAllocationCallbacks, VkDevice, VkSemaphore, VkSemaphoreCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateSemaphore

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkCreateShaderModule(3)

Name

vkCreateShaderModule - Creates a new shader module object

C Specification

To create a shader module, call:

```
VkResult vkCreateShaderModule(
    VkDevice                                    device,
    const VkShaderModuleCreateInfo*             pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkShaderModule*                             pShaderModule);
```

Parameters

- `device` is the logical device that creates the shader module.
- `pCreateInfo` parameter is a pointer to an instance of the `VkShaderModuleCreateInfo` structure.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.
- `pShaderModule` points to a `VkShaderModule` handle in which the resulting shader module object is returned.

Description

Once a shader module has been created, any entry points it contains **can** be used in pipeline shader stages as described in Compute Pipelines and Graphics Pipelines.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pCreateInfo` **must** be a pointer to a valid `VkShaderModuleCreateInfo` structure
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- `pShaderModule` **must** be a pointer to a `VkShaderModule` handle

Return Codes

Success

- VK_SUCCESS

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

See Also

VkAllocationCallbacks, VkDevice, VkShaderModule, VkShaderModuleCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkCreateShaderModule

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyBuffer(3)

Name

vkDestroyBuffer - Destroy a buffer object

C Specification

To destroy a buffer, call:

```
void vkDestroyBuffer(
    VkDevice                                    device,
    VkBuffer                                    buffer,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- All submitted commands that refer to `buffer`, either directly or via a `VkBufferView`, **must** have completed execution
- If `VkAllocationCallbacks` were provided when `buffer` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `buffer` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `buffer` is not VK_NULL_HANDLE, `buffer` **must** be a valid `VkBuffer` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkBuffer, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyBuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyBufferView(3)

Name

vkDestroyBufferView - Destroy a buffer view object

C Specification

To destroy a buffer view, call:

```
void vkDestroyBufferView(
    VkDevice                                    device,
    VkBufferView                                bufferView,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- All submitted commands that refer to `bufferView` **must** have completed execution
- If `VkAllocationCallbacks` were provided when `bufferView` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `bufferView` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `bufferView` is not VK_NULL_HANDLE, `bufferView` **must** be a valid `VkBufferView` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkBufferView, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyBufferView

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyCommandPool(3)

Name

vkDestroyCommandPool - Destroy a command pool object

C Specification

To destroy a command pool, call:

```
void vkDestroyCommandPool(
    VkDevice                                    device,
    VkCommandPool                               commandPool,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `device` is the logical device that destroys the command pool.
- `commandPool` is the handle of the command pool to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

Description

When a pool is destroyed, all command buffers allocated from the pool are freed.

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

- All `VkCommandBuffer` objects allocated from `commandPool` **must** not be in the pending state.
- If `VkAllocationCallbacks` were provided when `commandPool` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `commandPool` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `commandPool` is not VK_NULL_HANDLE, `commandPool` **must** be a valid `VkCommandPool` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- If `commandPool` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `commandPool` **must** be externally synchronized

See Also

VkAllocationCallbacks, VkCommandPool, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyCommandPool

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyDescriptorPool(3)

Name

vkDestroyDescriptorPool - Destroy a descriptor pool object

C Specification

To destroy a descriptor pool, call:

```
void vkDestroyDescriptorPool(
    VkDevice                                    device,
    VkDescriptorPool                            descriptorPool,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `device` is the logical device that destroys the descriptor pool.
- `descriptorPool` is the descriptor pool to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

Description

When a pool is destroyed, all descriptor sets allocated from the pool are implicitly freed and become invalid. Descriptor sets allocated from a given pool do not need to be freed before destroying that descriptor pool.

Valid Usage

- All submitted commands that refer to `descriptorPool` (via any allocated descriptor sets) **must** have completed execution
- If `VkAllocationCallbacks` were provided when `descriptorPool` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `descriptorPool` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `descriptorPool` is not VK_NULL_HANDLE, `descriptorPool` **must** be a valid `VkDescriptorPool` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- If `descriptorPool` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `descriptorPool` **must** be externally synchronized

See Also

VkAllocationCallbacks, VkDescriptorPool, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyDescriptorPool

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyDescriptorSetLayout(3)

Name

vkDestroyDescriptorSetLayout - Destroy a descriptor set layout object

C Specification

To destroy a descriptor set layout, call:

```
void vkDestroyDescriptorSetLayout(
    VkDevice                                    device,
    VkDescriptorSetLayout                       descriptorSetLayout,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- If `VkAllocationCallbacks` were provided when `descriptorSetLayout` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `descriptorSetLayout` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `descriptorSetLayout` is not VK_NULL_HANDLE, `descriptorSetLayout` **must** be a valid `VkDescriptorSetLayout` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkDescriptorSetLayout, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyDescriptorSetLayout

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyDevice(3)

Name

vkDestroyDevice - Destroy a logical device

C Specification

To destroy a device, call:

```
void vkDestroyDevice(
    VkDevice                                    device,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `device` is the logical device to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

Description

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 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

- All child objects created on `device` **must** have been destroyed prior to destroying `device`
- If `VkAllocationCallbacks` were provided when `device` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `device` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- If `device` is not `NULL`, `device` **must** be a valid `VkDevice` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure

Host Synchronization

- Host access to `device` **must** be externally synchronized

See Also

VkAllocationCallbacks, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyDevice

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyEvent(3)

Name

vkDestroyEvent - Destroy an event object

C Specification

To destroy an event, call:

```
void vkDestroyEvent(
    VkDevice                                    device,
    VkEvent                                     event,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- All submitted commands that refer to `event` **must** have completed execution
- If `VkAllocationCallbacks` were provided when `event` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `event` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `event` is not VK_NULL_HANDLE, `event` **must** be a valid `VkEvent` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkDevice, VkEvent

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyEvent

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyFence(3)

Name

vkDestroyFence - Destroy a fence object

C Specification

To destroy a fence, call:

```
void vkDestroyFence(
    VkDevice                                    device,
    VkFence                                     fence,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- All queue submission commands that refer to `fence` **must** have completed execution
- If `VkAllocationCallbacks` were provided when `fence` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `fence` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `fence` is not VK_NULL_HANDLE, `fence` **must** be a valid `VkFence` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkDevice, VkFence

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyFence

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyFramebuffer(3)

Name

vkDestroyFramebuffer - Destroy a framebuffer object

C Specification

To destroy a framebuffer, call:

```
void vkDestroyFramebuffer(
    VkDevice                                    device,
    VkFramebuffer                               framebuffer,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- All submitted commands that refer to `framebuffer` **must** have completed execution
- If `VkAllocationCallbacks` were provided when `framebuffer` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `framebuffer` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `framebuffer` is not VK_NULL_HANDLE, `framebuffer` **must** be a valid `VkFramebuffer` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkDevice, VkFramebuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyFramebuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyImage(3)

Name

vkDestroyImage - Destroy an image object

C Specification

To destroy an image, call:

```
void vkDestroyImage(
    VkDevice                                    device,
    VkImage                                     image,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- All submitted commands that refer to `image`, either directly or via a `VkImageView`, **must** have completed execution
- If `VkAllocationCallbacks` were provided when `image` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `image` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `image` is not VK_NULL_HANDLE, `image` **must** be a valid `VkImage` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkDevice, VkImage

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyImage

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyImageView(3)

Name

vkDestroyImageView - Destroy an image view object

C Specification

To destroy an image view, call:

```
void vkDestroyImageView(
    VkDevice                                    device,
    VkImageView                                 imageView,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- All submitted commands that refer to `imageView` **must** have completed execution
- If `VkAllocationCallbacks` were provided when `imageView` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `imageView` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `imageView` is not VK_NULL_HANDLE, `imageView` **must** be a valid `VkImageView` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkDevice, VkImageView

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyImageView

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyInstance(3)

Name

vkDestroyInstance - Destroy an instance of Vulkan

C Specification

To destroy an instance, call:

```
void vkDestroyInstance(
    VkInstance                                  instance,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

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

Description

Valid Usage

- All child objects created using `instance` **must** have been destroyed prior to destroying `instance`
- If `VkAllocationCallbacks` were provided when `instance` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `instance` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- If `instance` is not `NULL`, `instance` **must** be a valid `VkInstance` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure

Host Synchronization

- Host access to `instance` **must** be externally synchronized

See Also

VkAllocationCallbacks, VkInstance

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyInstance

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyPipeline(3)

Name

vkDestroyPipeline - Destroy a pipeline object

C Specification

To destroy a graphics or compute pipeline, call:

```
void vkDestroyPipeline(
    VkDevice                                    device,
    VkPipeline                                  pipeline,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- All submitted commands that refer to `pipeline` **must** have completed execution
- If `VkAllocationCallbacks` were provided when `pipeline` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `pipeline` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `pipeline` is not VK_NULL_HANDLE, `pipeline` **must** be a valid `VkPipeline` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkDevice, VkPipeline

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyPipeline

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyPipelineCache(3)

Name

vkDestroyPipelineCache - Destroy a pipeline cache object

C Specification

To destroy a pipeline cache, call:

```
void vkDestroyPipelineCache(
    VkDevice                                    device,
    VkPipelineCache                             pipelineCache,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- If `VkAllocationCallbacks` were provided when `pipelineCache` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `pipelineCache` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `pipelineCache` is not VK_NULL_HANDLE, `pipelineCache` **must** be a valid `VkPipelineCache` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkDevice, VkPipelineCache

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyPipelineCache

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyPipelineLayout(3)

Name

vkDestroyPipelineLayout - Destroy a pipeline layout object

C Specification

To destroy a pipeline layout, call:

```
void vkDestroyPipelineLayout(
    VkDevice                                    device,
    VkPipelineLayout                            pipelineLayout,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- If `VkAllocationCallbacks` were provided when `pipelineLayout` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `pipelineLayout` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `pipelineLayout` is not VK_NULL_HANDLE, `pipelineLayout` **must** be a valid `VkPipelineLayout` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkDevice, VkPipelineLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyPipelineLayout

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyQueryPool(3)

Name

vkDestroyQueryPool - Destroy a query pool object

C Specification

To destroy a query pool, call:

```
void vkDestroyQueryPool(
    VkDevice                                    device,
    VkQueryPool                                 queryPool,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `device` is the logical device that destroys the query pool.
- `queryPool` is the query pool to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

Description

Valid Usage

- All submitted commands that refer to `queryPool` **must** have completed execution
- If `VkAllocationCallbacks` were provided when `queryPool` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `queryPool` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `queryPool` is not VK_NULL_HANDLE, `queryPool` **must** be a valid `VkQueryPool` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- If `queryPool` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `queryPool` **must** be externally synchronized

See Also

VkAllocationCallbacks, VkDevice, VkQueryPool

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyQueryPool

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyRenderPass(3)

Name

vkDestroyRenderPass - Destroy a render pass object

C Specification

To destroy a render pass, call:

```
void vkDestroyRenderPass(
    VkDevice                                    device,
    VkRenderPass                                renderPass,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- All submitted commands that refer to `renderPass` **must** have completed execution
- If `VkAllocationCallbacks` were provided when `renderPass` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `renderPass` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `renderPass` is not VK_NULL_HANDLE, `renderPass` **must** be a valid `VkRenderPass` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkDevice, VkRenderPass

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyRenderPass

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroySampler(3)

Name

vkDestroySampler - Destroy a sampler object

C Specification

To destroy a sampler, call:

```
void vkDestroySampler(
    VkDevice                                    device,
    VkSampler                                   sampler,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- All submitted commands that refer to `sampler` **must** have completed execution
- If `VkAllocationCallbacks` were provided when `sampler` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `sampler` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `sampler` is not VK_NULL_HANDLE, `sampler` **must** be a valid `VkSampler` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkDevice, VkSampler

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroySampler

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroySemaphore(3)

Name

vkDestroySemaphore - Destroy a semaphore object

C Specification

To destroy a semaphore, call:

```
void vkDestroySemaphore(
    VkDevice                                    device,
    VkSemaphore                                 semaphore,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `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.

Description

Valid Usage

- All submitted batches that refer to `semaphore` **must** have completed execution
- If `VkAllocationCallbacks` were provided when `semaphore` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `semaphore` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `semaphore` is not VK_NULL_HANDLE, `semaphore` **must** be a valid `VkSemaphore` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- 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

See Also

VkAllocationCallbacks, VkDevice, VkSemaphore

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroySemaphore

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDestroyShaderModule(3)

Name

vkDestroyShaderModule - Destroy a shader module module

C Specification

To destroy a shader module, call:

```
void vkDestroyShaderModule(
    VkDevice                                    device,
    VkShaderModule                              shaderModule,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `device` is the logical device that destroys the shader module.
- `shaderModule` is the handle of the shader module to destroy.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

Description

A shader module **can** be destroyed while pipelines created using its shaders are still in use.

Valid Usage

- If `VkAllocationCallbacks` were provided when `shaderModule` was created, a compatible set of callbacks **must** be provided here
- If no `VkAllocationCallbacks` were provided when `shaderModule` was created, `pAllocator` **must** be `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `shaderModule` is not VK_NULL_HANDLE, `shaderModule` **must** be a valid `VkShaderModule` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- If `shaderModule` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `shaderModule` **must** be externally synchronized

See Also

VkAllocationCallbacks, VkDevice, VkShaderModule

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDestroyShaderModule

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkDeviceWaitIdle(3)

Name

vkDeviceWaitIdle - Wait for a device to become idle

C Specification

To wait on the host for the completion of outstanding queue operations for all queues on a given logical device, call:

```
VkResult vkDeviceWaitIdle(
    VkDevice                                    device);
```

Parameters

- `device` is the logical device to idle.

Description

`vkDeviceWaitIdle` is equivalent to calling `vkQueueWaitIdle` for all queues owned by `device`.

Valid Usage (Implicit)

- `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_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`

See Also

VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkDeviceWaitIdle

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkEndCommandBuffer(3)

Name

vkEndCommandBuffer - Finish recording a command buffer

C Specification

To complete recording of a command buffer, call:

```
VkResult vkEndCommandBuffer(
    VkCommandBuffer                             commandBuffer);
```

Parameters

- `commandBuffer` is the command buffer to complete recording.

Description

If there was an error during recording, the application will be notified by an unsuccessful return code returned by `vkEndCommandBuffer`. If the application wishes to further use the command buffer, the command buffer **must** be reset. The command buffer **must** have been in the recording state, and is moved to the executable state.

Valid Usage

- `commandBuffer` **must** be in the recording state.
- If `commandBuffer` is a primary command buffer, there **must** not be an active render pass instance
- All queries made active during the recording of `commandBuffer` **must** have been made inactive

Valid Usage (Implicit)

- `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_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkEndCommandBuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkEnumerateDeviceExtensionProperties(3)

Name

vkEnumerateDeviceExtensionProperties - Returns properties of available physical device extensions

C Specification

To query the extensions available to a given physical device, call:

```
VkResult vkEnumerateDeviceExtensionProperties(
    VkPhysicalDevice                            physicalDevice,
    const char*                                 pLayerName,
    uint32_t*                                   pPropertyCount,
    VkExtensionProperties*                      pProperties);
```

Parameters

- `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.

Description

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.

Valid Usage (Implicit)

- `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- If `pLayerName` is not `NULL`, `pLayerName` **must** be a null-terminated UTF-8 string
- `pPropertyCount` **must** be a pointer to a `uint32_t` value
- If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a pointer to an array of `pPropertyCount` `VkExtensionProperties` structures

Return Codes

Success

- `VK_SUCCESS`
- `VK_INCOMPLETE`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_LAYER_NOT_PRESENT`

See Also

VkExtensionProperties, VkPhysicalDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkEnumerateDeviceExtensionProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkEnumerateDeviceLayerProperties(3)

Name

vkEnumerateDeviceLayerProperties - Returns properties of available physical device layers

C Specification

To enumerate device layers, call:

```
VkResult vkEnumerateDeviceLayerProperties(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pPropertyCount,
    VkLayerProperties*                          pProperties);
```

Parameters

- `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.

Description

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 user 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. If `pPropertyCount` is smaller than the number of layers available, `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available layer properties were returned.

The list of layers enumerated by `vkEnumerateDeviceLayerProperties` **must** be exactly the sequence of layers enabled for the instance. The members of `VkLayerProperties` for each enumerated layer **must** be the same as the properties when the layer was enumerated by `vkEnumerateInstanceLayerProperties`.

Valid Usage (Implicit)

- `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- `pPropertyCount` **must** be a pointer to a `uint32_t` value
- If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a pointer to an array of `pPropertyCount` `VkLayerProperties` structures

Return Codes

Success

- `VK_SUCCESS`
- `VK_INCOMPLETE`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkLayerProperties, VkPhysicalDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkEnumerateDeviceLayerProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkEnumerateInstanceExtensionProperties(3)

Name

vkEnumerateInstanceExtensionProperties - Returns up to requested number of global extension properties

C Specification

To query the available instance extensions, call:

```
VkResult vkEnumerateInstanceExtensionProperties(
    const char*                                 pLayerName,
    uint32_t*                                   pPropertyCount,
    VkExtensionProperties*                      pProperties);
```

Parameters

- `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.

Description

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 user 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. If `pPropertyCount` is smaller than the number of extensions available, `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.

Valid Usage (Implicit)

- If `pLayerName` is not `NULL`, `pLayerName` **must** be a null-terminated UTF-8 string
- `pPropertyCount` **must** be a pointer to a `uint32_t` value
- If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a pointer to an array of `pPropertyCount` `VkExtensionProperties` structures

Return Codes

Success

- `VK_SUCCESS`
- `VK_INCOMPLETE`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_LAYER_NOT_PRESENT`

See Also

VkExtensionProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkEnumerateInstanceExtensionProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkEnumerateInstanceLayerProperties(3)

Name

vkEnumerateInstanceLayerProperties - Returns up to requested number of global layer properties

C Specification

To query the available layers, call:

```
VkResult vkEnumerateInstanceLayerProperties(
    uint32_t*                                   pPropertyCount,
    VkLayerProperties*                          pProperties);
```

Parameters

- `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.

Description

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 user 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. If `pPropertyCount` is smaller than the number of layers available, `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available layer 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)

- `pPropertyCount` **must** be a pointer to a `uint32_t` value
- If the value referenced by `pPropertyCount` is not `0`, and `pProperties` is not `NULL`, `pProperties` **must** be a pointer to an array of `pPropertyCount` `VkLayerProperties` structures

Return Codes

Success

- `VK_SUCCESS`
- `VK_INCOMPLETE`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkLayerProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkEnumerateInstanceLayerProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkEnumeratePhysicalDevices(3)

Name

vkEnumeratePhysicalDevices - Enumerates the physical devices accessible to a Vulkan instance

C Specification

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

```
VkResult vkEnumeratePhysicalDevices(
    VkInstance                                  instance,
    uint32_t*                                   pPhysicalDeviceCount,
    VkPhysicalDevice*                           pPhysicalDevices);
```

Parameters

- `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.

Description

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 user 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. If `pPhysicalDeviceCount` is smaller than the number of physical devices available, `VK_INCOMPLETE` will be returned instead of `VK_SUCCESS`, to indicate that not all the available physical devices were returned.

Valid Usage (Implicit)

- `instance` **must** be a valid `VkInstance` handle
- `pPhysicalDeviceCount` **must** be a pointer to a `uint32_t` value
- If the value referenced by `pPhysicalDeviceCount` is not `0`, and `pPhysicalDevices` is not `NULL`, `pPhysicalDevices` **must** be a pointer to an array of `pPhysicalDeviceCount` `VkPhysicalDevice` handles

Return Codes

Success

- `VK_SUCCESS`
- `VK_INCOMPLETE`

Failure

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

See Also

VkInstance, VkPhysicalDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkEnumeratePhysicalDevices

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkFlushMappedMemoryRanges(3)

Name

vkFlushMappedMemoryRanges - Flush mapped memory ranges

C Specification

To flush ranges of non-coherent memory from the host caches, call:

```
VkResult vkFlushMappedMemoryRanges(
    VkDevice                                    device,
    uint32_t                                    memoryRangeCount,
    const VkMappedMemoryRange*                  pMemoryRanges);
```

Parameters

- `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.

Description

`vkFlushMappedMemoryRanges` guarantees that host writes to the memory ranges described by `pMemoryRanges` **can** be made available to device access, via availability operations from the `VK_ACCESS_HOST_WRITE_BIT` access type.

Unmapping non-coherent memory does not implicitly flush the 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).

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pMemoryRanges` **must** be a pointer to an array of `memoryRangeCount` valid `VkMappedMemoryRange` structures
- `memoryRangeCount` **must** be greater than `0`

Return Codes

Success

- VK_SUCCESS

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

See Also

VkDevice, VkMappedMemoryRange

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkFlushMappedMemoryRanges

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkFreeCommandBuffers(3)

Name

vkFreeCommandBuffers - Free command buffers

C Specification

To free command buffers, call:

```
void vkFreeCommandBuffers(
    VkDevice                                    device,
    VkCommandPool                               commandPool,
    uint32_t                                    commandBufferCount,
    const VkCommandBuffer*                      pCommandBuffers);
```

Parameters

- `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 an array of handles of command buffers to free.

Description

Any primary command buffer that is in the recording or executable state and has any element of `pCommandBuffers` recorded into it, becomes invalid.

Valid Usage

- All elements of `pCommandBuffers` **must** not be in the pending state
- `pCommandBuffers` **must** be a pointer to an array of `commandBufferCount` `VkCommandBuffer` handles, each element of which **must** either be a valid handle or `NULL`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `commandPool` **must** be a valid `VkCommandPool` handle
- `commandBufferCount` **must** be greater than `0`
- `commandPool` **must** have been created, allocated, or retrieved from `device`
- 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

See Also

VkCommandBuffer, VkCommandPool, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkFreeCommandBuffers

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkFreeDescriptorSets(3)

Name

vkFreeDescriptorSets - Free one or more descriptor sets

C Specification

To free allocated descriptor sets, call:

```
VkResult vkFreeDescriptorSets(
    VkDevice                                    device,
    VkDescriptorPool                            descriptorPool,
    uint32_t                                    descriptorSetCount,
    const VkDescriptorSet*                      pDescriptorSets);
```

Parameters

- `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 an array of handles to `VkDescriptorSet` objects.

Description

After a successful call to `vkFreeDescriptorSets`, all descriptor sets in `pDescriptorSets` are invalid.

Valid Usage

- All submitted commands that refer to any element of `pDescriptorSets` **must** have completed execution
- `pDescriptorSets` **must** be a pointer to an array of `descriptorSetCount` `VkDescriptorSet` handles, each element of which **must** either be a valid handle or VK_NULL_HANDLE
- Each valid handle in `pDescriptorSets` **must** have been allocated from `descriptorPool`
- `descriptorPool` **must** have been created with the `VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT` flag

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `descriptorPool` **must** be a valid `VkDescriptorPool` handle
- `descriptorSetCount` **must** be greater than `0`
- `descriptorPool` **must** have been created, allocated, or retrieved from `device`
- 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_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkDescriptorPool, VkDescriptorSet, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkFreeDescriptorSets

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkFreeMemory(3)

Name

vkFreeMemory - Free GPU memory

C Specification

To free a memory object, call:

```
void vkFreeMemory(
    VkDevice                                    device,
    VkDeviceMemory                              memory,
    const VkAllocationCallbacks*                pAllocator);
```

Parameters

- `device` is the logical device that owns the memory.
- `memory` is the `VkDeviceMemory` object to be freed.
- `pAllocator` controls host memory allocation as described in the Memory Allocation chapter.

Description

Before freeing a memory object, an application **must** ensure the memory object is no longer in use by the device—for example by command buffers queued for execution. The memory **can** remain bound to images or buffers at the time the memory object is freed, but any further use of them (on host or device) for anything other than destroying those objects will result in undefined behavior. If there are still any bound images or buffers, the memory **may** not be immediately released by the implementation, but **must** be released by the time all bound images and buffers have been destroyed. Once memory is released, it is returned to the heap from which it was allocated.

How memory objects are bound to Images and Buffers is described in detail in the Resource Memory Association section.

If a memory object is mapped at the time it is freed, it is implicitly unmapped.

> *Note*
>
> As described below, host writes are not implicitly flushed when the memory object is unmapped, but the implementation **must** guarantee that writes that have not been flushed do not affect any other memory.

Valid Usage

- All submitted commands that refer to `memory` (via images or buffers) **must** have completed execution

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `memory` is not VK_NULL_HANDLE, `memory` **must** be a valid `VkDeviceMemory` handle
- If `pAllocator` is not `NULL`, `pAllocator` **must** be a pointer to a valid `VkAllocationCallbacks` structure
- If `memory` is a valid handle, it **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `memory` **must** be externally synchronized

See Also

VkAllocationCallbacks, VkDevice, VkDeviceMemory

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkFreeMemory

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetBufferMemoryRequirements(3)

Name

vkGetBufferMemoryRequirements - Returns the memory requirements for specified Vulkan object

C Specification

To determine the memory requirements for a buffer resource, call:

```
void vkGetBufferMemoryRequirements(
    VkDevice                                    device,
    VkBuffer                                    buffer,
    VkMemoryRequirements*                       pMemoryRequirements);
```

Parameters

- `device` is the logical device that owns the buffer.
- `buffer` is the buffer to query.
- `pMemoryRequirements` points to an instance of the VkMemoryRequirements structure in which the memory requirements of the buffer object are returned.

Description

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `buffer` **must** be a valid `VkBuffer` handle
- `pMemoryRequirements` **must** be a pointer to a `VkMemoryRequirements` structure
- `buffer` **must** have been created, allocated, or retrieved from `device`

See Also

VkBuffer, VkDevice, VkMemoryRequirements

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetBufferMemoryRequirements

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetDeviceMemoryCommitment(3)

Name

vkGetDeviceMemoryCommitment - Query the current commitment for a VkDeviceMemory

C Specification

To determine the amount of lazily-allocated memory that is currently committed for a memory object, call:

```
void vkGetDeviceMemoryCommitment(
    VkDevice                                    device,
    VkDeviceMemory                              memory,
    VkDeviceSize*                               pCommittedMemoryInBytes);
```

Parameters

- `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.

Description

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

- `memory` **must** have been created with a memory type that reports `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `memory` **must** be a valid `VkDeviceMemory` handle
- `pCommittedMemoryInBytes` **must** be a pointer to a `VkDeviceSize` value
- `memory` **must** have been created, allocated, or retrieved from `device`

See Also

VkDevice, VkDeviceMemory, `VkDeviceSize`

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetDeviceMemoryCommitment

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetDeviceProcAddr(3)

Name

vkGetDeviceProcAddr - Return a function pointer for a command

C Specification

In order to support systems with multiple Vulkan implementations comprising heterogeneous collections of hardware and software, the function pointers returned by `vkGetInstanceProcAddr` **may** point to dispatch code, which calls a different real implementation for different `VkDevice` objects (and objects created from them). The overhead of this internal dispatch **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 with the command:

```
PFN_vkVoidFunction vkGetDeviceProcAddr(
    VkDevice                                    device,
    const char*                                 pName);
```

Parameters

The table below defines the various use cases for `vkGetDeviceProcAddr` and expected return value for each case.

Description

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

Table 1. vkGetDeviceProcAddr behavior

`device`	`pName`	**return value**
`NULL`	*	undefined
invalid device	*	undefined
device	`NULL`	undefined
device	core Vulkan command	fp[1]
device	enabled extension commands	fp[1]
device	* (any `pName` not covered above)	`NULL`

1

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)

- `device` **must** be a valid `VkDevice` handle
- `pName` **must** be a null-terminated UTF-8 string

See Also

PFN_vkVoidFunction, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetDeviceProcAddr

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetDeviceQueue(3)

Name

vkGetDeviceQueue - Get a queue handle from a device

C Specification

To retrieve a handle to a VkQueue object, call:

```
void vkGetDeviceQueue(
    VkDevice                                    device,
    uint32_t                                    queueFamilyIndex,
    uint32_t                                    queueIndex,
    VkQueue*                                    pQueue);
```

Parameters

- `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.

Description

Valid Usage

- `queueFamilyIndex` **must** be one of the queue family indices specified when `device` was created, via the `VkDeviceQueueCreateInfo` structure
- `queueIndex` **must** be less than the number of queues created for the specified queue family index when `device` was created, via the `queueCount` member of the `VkDeviceQueueCreateInfo` structure

Valid Usage (Implicit)

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

See Also

VkDevice, VkQueue

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetDeviceQueue

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetEventStatus(3)

Name

vkGetEventStatus - Retrieve the status of an event object

C Specification

To query the state of an event from the host, call:

```
VkResult vkGetEventStatus(
    VkDevice                                    device,
    VkEvent                                     event);
```

Parameters

- `device` is the logical device that owns the event.
- `event` is the handle of the event to query.

Description

Upon success, `vkGetEventStatus` returns the state of the event object with the following return codes:

Table 2. 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.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `event` **must** be a valid `VkEvent` handle
- `event` **must** have been created, allocated, or retrieved from `device`

Return Codes

Success

- VK_EVENT_SET
- VK_EVENT_RESET

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST

See Also

VkDevice, VkEvent

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetEventStatus

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetFenceStatus(3)

Name

vkGetFenceStatus - Return the status of a fence

C Specification

To query the status of a fence from the host, call:

```
VkResult vkGetFenceStatus(
    VkDevice                                    device,
    VkFence                                     fence);
```

Parameters

- `device` is the logical device that owns the fence.
- `fence` is the handle of the fence to query.

Description

Upon success, `vkGetFenceStatus` returns the status of the fence object, with the following return codes:

Table 3. Fence Object Status Codes

Status	Meaning
`VK_SUCCESS`	The fence specified by `fence` is signaled.
`VK_NOT_READY`	The fence specified by `fence` is unsignaled.
`VK_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_DEVICE_LOST`.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `fence` **must** be a valid `VkFence` handle
- `fence` **must** have been created, allocated, or retrieved from `device`

Return Codes

Success

- `VK_SUCCESS`
- `VK_NOT_READY`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`

See Also

VkDevice, VkFence

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetFenceStatus

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetImageMemoryRequirements(3)

Name

vkGetImageMemoryRequirements - Returns the memory requirements for specified Vulkan object

C Specification

To determine the memory requirements for an image resource, call:

```
void vkGetImageMemoryRequirements(
    VkDevice                                    device,
    VkImage                                     image,
    VkMemoryRequirements*                       pMemoryRequirements);
```

Parameters

- `device` is the logical device that owns the image.
- `image` is the image to query.
- `pMemoryRequirements` points to an instance of the VkMemoryRequirements structure in which the memory requirements of the image object are returned.

Description

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `image` **must** be a valid `VkImage` handle
- `pMemoryRequirements` **must** be a pointer to a `VkMemoryRequirements` structure
- `image` **must** have been created, allocated, or retrieved from `device`

See Also

VkDevice, VkImage, VkMemoryRequirements

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetImageMemoryRequirements

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetImageSparseMemoryRequirements(3)

Name

vkGetImageSparseMemoryRequirements - Query the memory requirements for a sparse image

C Specification

To query sparse memory requirements for an image, call:

```
void vkGetImageSparseMemoryRequirements(
    VkDevice                                    device,
    VkImage                                     image,
    uint32_t*                                   pSparseMemoryRequirementCount,
    VkSparseImageMemoryRequirements*            pSparseMemoryRequirements);
```

Parameters

- `device` is the logical device that owns the image.
- `image` is the `VkImage` object to get the memory requirements for.
- `pSparseMemoryRequirementCount` is a pointer to an integer related to the number of sparse memory requirements available or queried, as described below.
- `pSparseMemoryRequirements` is either `NULL` or a pointer to an array of `VkSparseImageMemoryRequirements` structures.

Description

If `pSparseMemoryRequirements` is `NULL`, then the number of sparse memory requirements available is returned in `pSparseMemoryRequirementCount`. Otherwise, `pSparseMemoryRequirementCount` **must** point to a variable set by the user to the number of elements in the `pSparseMemoryRequirements` array, and on return the variable is overwritten with the number of structures actually written to `pSparseMemoryRequirements`. If `pSparseMemoryRequirementCount` is less than the number of sparse memory requirements available, at most `pSparseMemoryRequirementCount` structures will be written.

If the image was not created with `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` then `pSparseMemoryRequirementCount` will be set to zero and `pSparseMemoryRequirements` will not be written to.

> *Note*
>
> It is legal for an implementation to report a larger value in `VkMemoryRequirements::size` than would be obtained by adding together memory sizes for all `VkSparseImageMemoryRequirements` returned by `vkGetImageSparseMemoryRequirements`. This **may** occur when the hardware requires unused padding in the address range describing the resource.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `image` **must** be a valid `VkImage` handle
- `pSparseMemoryRequirementCount` **must** be a pointer to a `uint32_t` value
- If the value referenced by `pSparseMemoryRequirementCount` is not `0`, and `pSparseMemoryRequirements` is not `NULL`, `pSparseMemoryRequirements` **must** be a pointer to an array of `pSparseMemoryRequirementCount` `VkSparseImageMemoryRequirements` structures
- `image` **must** have been created, allocated, or retrieved from `device`

See Also

VkDevice, VkImage, VkSparseImageMemoryRequirements

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetImageSparseMemoryRequirements

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetImageSubresourceLayout(3)

Name

vkGetImageSubresourceLayout - Retrieve information about an image subresource

C Specification

To query the host access layout of an image subresource, for an image created with linear tiling, call:

```
void vkGetImageSubresourceLayout(
    VkDevice                                    device,
    VkImage                                     image,
    const VkImageSubresource*                   pSubresource,
    VkSubresourceLayout*                        pLayout);
```

Parameters

- `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 for the image subresource.
- `pLayout` points to a VkSubresourceLayout structure in which the layout is returned.

Description

vkGetImageSubresourceLayout is invariant for the lifetime of a single image.

Valid Usage

- `image` **must** have been created with `tiling` equal to `VK_IMAGE_TILING_LINEAR`
- The `aspectMask` member of `pSubresource` **must** only have a single bit set

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `image` **must** be a valid `VkImage` handle
- `pSubresource` **must** be a pointer to a valid `VkImageSubresource` structure
- `pLayout` **must** be a pointer to a `VkSubresourceLayout` structure
- `image` **must** have been created, allocated, or retrieved from `device`

See Also

VkDevice, VkImage, VkImageSubresource, VkSubresourceLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetImageSubresourceLayout

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetInstanceProcAddr(3)

Name

vkGetInstanceProcAddr - Return a function pointer for a command

C Specification

Vulkan commands are not necessarily exposed statically on a platform. Function pointers for all Vulkan commands **can** be obtained with the command:

```
PFN_vkVoidFunction vkGetInstanceProcAddr(
    VkInstance                                  instance,
    const char*                                 pName);
```

Parameters

- `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.

Description

`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. Loaders are encouraged to export function symbols for all other core Vulkan commands as well; if this is done, then applications that use only the core Vulkan commands have no need to use `vkGetInstanceProcAddr`.

The table below defines the various use cases for `vkGetInstanceProcAddr` and expected return value ("fp" is function pointer) for each case.

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

Table 4. vkGetInstanceProcAddr behavior

`instance`	`pName`	**return value**
*	`NULL`	undefined
invalid instance	*	undefined
`NULL`	vkEnumerateInstanceExtensionProperties	fp
`NULL`	vkEnumerateInstanceLayerProperties	fp
`NULL`	vkCreateInstance	fp

instance	pName	**return value**
NULL	* (any pName not covered above)	NULL
instance	core Vulkan command	fp[1]
instance	enabled instance extension commands for instance	fp[1]
instance	available device extension[2] commands for instance	fp[1]
instance	* (any pName not covered above)	NULL

1

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.

2

An "available extension" is an extension function supported by any of the loader, driver or layer.

Valid Usage (Implicit)

- If instance is not NULL, instance **must** be a valid VkInstance handle
- pName **must** be a null-terminated UTF-8 string

See Also

PFN_vkVoidFunction, VkInstance

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetInstanceProcAddr

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetPhysicalDeviceFeatures(3)

Name

vkGetPhysicalDeviceFeatures - Reports capabilities of a physical device

C Specification

To query supported features, call:

```
void vkGetPhysicalDeviceFeatures(
    VkPhysicalDevice                            physicalDevice,
    VkPhysicalDeviceFeatures*                   pFeatures);
```

Parameters

- `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` indicates that the feature is supported on this physical device, and `VK_FALSE` indicates that the feature is not supported.

Description

Valid Usage (Implicit)

- `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- `pFeatures` **must** be a pointer to a `VkPhysicalDeviceFeatures` structure

See Also

VkPhysicalDevice, VkPhysicalDeviceFeatures

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetPhysicalDeviceFeatures

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetPhysicalDeviceFormatProperties(3)

Name

vkGetPhysicalDeviceFormatProperties - Lists physical device's format capabilities

C Specification

To query supported format features which are properties of the physical device, call:

```
void vkGetPhysicalDeviceFormatProperties(
    VkPhysicalDevice                            physicalDevice,
    VkFormat                                    format,
    VkFormatProperties*                         pFormatProperties);
```

Parameters

- `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.

Description

Valid Usage (Implicit)

- `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- `format` **must** be a valid VkFormat value
- `pFormatProperties` **must** be a pointer to a `VkFormatProperties` structure

See Also

VkFormat, VkFormatProperties, VkPhysicalDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetPhysicalDeviceFormatProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetPhysicalDeviceImageFormatProperties(3)

Name

vkGetPhysicalDeviceImageFormatProperties - Lists physical device's image format capabilities

C Specification

To query additional capabilities specific to image types, call:

```
VkResult vkGetPhysicalDeviceImageFormatProperties(
    VkPhysicalDevice                            physicalDevice,
    VkFormat                                    format,
    VkImageType                                 type,
    VkImageTiling                               tiling,
    VkImageUsageFlags                           usage,
    VkImageCreateFlags                          flags,
    VkImageFormatProperties*                    pImageFormatProperties);
```

Parameters

- `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` points to an instance of the VkImageFormatProperties structure in which capabilities are returned.

Description

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.

Valid Usage (Implicit)

- physicalDevice **must** be a valid VkPhysicalDevice handle
- format **must** be a valid VkFormat value
- type **must** be a valid VkImageType value
- tiling **must** be a valid VkImageTiling value
- usage **must** be a valid combination of VkImageUsageFlagBits values
- usage **must** not be 0
- flags **must** be a valid combination of VkImageCreateFlagBits values
- pImageFormatProperties **must** be a pointer to a VkImageFormatProperties structure

Return Codes

Success

- VK_SUCCESS

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_FORMAT_NOT_SUPPORTED

See Also

VkFormat, VkImageCreateFlags, VkImageFormatProperties, VkImageTiling, VkImageType, VkImageUsageFlags, VkPhysicalDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetPhysicalDeviceImageFormatProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetPhysicalDeviceMemoryProperties(3)

Name

vkGetPhysicalDeviceMemoryProperties - Reports memory information for the specified physical device

C Specification

To query memory properties, call:

```
void vkGetPhysicalDeviceMemoryProperties(
    VkPhysicalDevice                            physicalDevice,
    VkPhysicalDeviceMemoryProperties*           pMemoryProperties);
```

Parameters

- `physicalDevice` is the handle to the device to query.
- `pMemoryProperties` points to an instance of `VkPhysicalDeviceMemoryProperties` structure in which the properties are returned.

Description

Valid Usage (Implicit)

- `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- `pMemoryProperties` **must** be a pointer to a `VkPhysicalDeviceMemoryProperties` structure

See Also

VkPhysicalDevice, VkPhysicalDeviceMemoryProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetPhysicalDeviceMemoryProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetPhysicalDeviceProperties(3)

Name

vkGetPhysicalDeviceProperties - Returns properties of a physical device

C Specification

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

```
void vkGetPhysicalDeviceProperties(
    VkPhysicalDevice                            physicalDevice,
    VkPhysicalDeviceProperties*                 pProperties);
```

Parameters

- `physicalDevice` is the handle to the physical device whose properties will be queried.
- `pProperties` points to an instance of the VkPhysicalDeviceProperties structure, that will be filled with returned information.

Description

Valid Usage (Implicit)

- `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- `pProperties` **must** be a pointer to a `VkPhysicalDeviceProperties` structure

See Also

VkPhysicalDevice, VkPhysicalDeviceProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetPhysicalDeviceProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetPhysicalDeviceQueueFamilyProperties(3)

Name

vkGetPhysicalDeviceQueueFamilyProperties - Reports properties of the queues of the specified physical device

C Specification

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

```
void vkGetPhysicalDeviceQueueFamilyProperties(
    VkPhysicalDevice                            physicalDevice,
    uint32_t*                                   pQueueFamilyPropertyCount,
    VkQueueFamilyProperties*                    pQueueFamilyProperties);
```

Parameters

- `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.

Description

If `pQueueFamilyProperties` is `NULL`, then the number of queue families available is returned in `pQueueFamilyPropertyCount`. Otherwise, `pQueueFamilyPropertyCount` **must** point to a variable set by the user 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)

- `physicalDevice` **must** be a valid `VkPhysicalDevice` handle
- `pQueueFamilyPropertyCount` **must** be a pointer to a `uint32_t` value
- If the value referenced by `pQueueFamilyPropertyCount` is not `0`, and `pQueueFamilyProperties` is not `NULL`, `pQueueFamilyProperties` **must** be a pointer to an array of `pQueueFamilyPropertyCount` `VkQueueFamilyProperties` structures

See Also

VkPhysicalDevice, VkQueueFamilyProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetPhysicalDeviceQueueFamilyProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetPhysicalDeviceSparseImageFormatProperties(3)

Name

vkGetPhysicalDeviceSparseImageFormatProperties - Retrieve properties of an image format applied to sparse images

C Specification

`vkGetPhysicalDeviceSparseImageFormatProperties` returns an array of VkSparseImageFormatProperties. Each element will describe properties for one set of image aspects that are bound simultaneously in the image. This is usually one element for each aspect in the image, but for interleaved depth/stencil images there is only one element describing the combined aspects.

```
void vkGetPhysicalDeviceSparseImageFormatProperties(
    VkPhysicalDevice                            physicalDevice,
    VkFormat                                    format,
    VkImageType                                 type,
    VkSampleCountFlagBits                       samples,
    VkImageUsageFlags                           usage,
    VkImageTiling                               tiling,
    uint32_t*                                   pPropertyCount,
    VkSparseImageFormatProperties*              pProperties);
```

Parameters

- `physicalDevice` is the physical device from which to query the sparse image capabilities.
- `format` is the image format.
- `type` is the dimensionality of image.
- `samples` is the number of samples per pixel as defined in VkSampleCountFlagBits.
- `usage` is a bitmask describing the intended usage of the image.
- `tiling` is the tiling arrangement of the data elements in memory.
- `pPropertyCount` is a pointer to an integer related to the number of sparse format properties available or queried, as described below.
- `pProperties` is either `NULL` or a pointer to an array of VkSparseImageFormatProperties structures.

Description

If `pProperties` is `NULL`, then the number of sparse format properties available is returned in `pPropertyCount`. Otherwise, `pPropertyCount` **must** point to a variable set by the user 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 sparse format properties available, at most pPropertyCount structures will be written.

If VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT is not supported for the given arguments, pPropertyCount will be set to zero upon return, and no data will be written to pProperties.

Multiple aspects are returned for depth/stencil images that are implemented as separate planes by the implementation. The depth and stencil data planes each have unique VkSparseImageFormatProperties data.

Depth/stencil images with depth and stencil data interleaved into a single plane will return a single VkSparseImageFormatProperties structure with the aspectMask set to VK_IMAGE_ASPECT_DEPTH_BIT | VK_IMAGE_ASPECT_STENCIL_BIT.

Valid Usage

- samples **must** be a bit value that is set in VkImageFormatProperties::sampleCounts returned by vkGetPhysicalDeviceImageFormatProperties with format, type, tiling, and usage equal to those in this command and flags equal to the value that is set in VkImageCreateInfo::flags when the image is created

Valid Usage (Implicit)

- physicalDevice **must** be a valid VkPhysicalDevice handle
- format **must** be a valid VkFormat value
- type **must** be a valid VkImageType value
- samples **must** be a valid VkSampleCountFlagBits value
- usage **must** be a valid combination of VkImageUsageFlagBits values
- usage **must** not be 0
- tiling **must** be a valid VkImageTiling value
- pPropertyCount **must** be a pointer to a uint32_t value
- If the value referenced by pPropertyCount is not 0, and pProperties is not NULL, pProperties **must** be a pointer to an array of pPropertyCount VkSparseImageFormatProperties structures

See Also

VkFormat, VkImageTiling, VkImageType, VkImageUsageFlags, VkPhysicalDevice, VkSampleCountFlagBits, VkSparseImageFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetPhysicalDeviceSparseImageFormatProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetPipelineCacheData(3)

Name

vkGetPipelineCacheData - Get the data store from a pipeline cache

C Specification

Data **can** be retrieved from a pipeline cache object using the command:

```
VkResult vkGetPipelineCacheData(
    VkDevice                                    device,
    VkPipelineCache                             pipelineCache,
    size_t*                                     pDataSize,
    void*                                       pData);
```

Parameters

- `device` is the logical device that owns the pipeline cache.
- `pipelineCache` is the pipeline cache to retrieve data from.
- `pDataSize` is a pointer to a value related to the amount of data in the pipeline cache, as described below.
- `pData` is either `NULL` or a pointer to a buffer.

Description

If `pData` is `NULL`, then the maximum size of the data that **can** be retrieved from the pipeline cache, in bytes, is returned in `pDataSize`. Otherwise, `pDataSize` **must** point to a variable set by the user to the size of the buffer, in bytes, pointed to by `pData`, and on return the variable is overwritten with the amount of data actually written to `pData`.

If `pDataSize` is less than the maximum size that **can** be retrieved by the pipeline cache, at most `pDataSize` bytes will be written to `pData`, and `vkGetPipelineCacheData` will return `VK_INCOMPLETE`. Any data written to `pData` is valid and **can** be provided as the `pInitialData` member of the `VkPipelineCacheCreateInfo` structure passed to `vkCreatePipelineCache`.

Two calls to `vkGetPipelineCacheData` with the same parameters **must** retrieve the same data unless a command that modifies the contents of the cache is called between them.

Applications **can** store the data retrieved from the pipeline cache, and use these data, possibly in a future run of the application, to populate new pipeline cache objects. The results of pipeline compiles, however, **may** depend on the vendor ID, device ID, driver version, and other details of the device. To enable applications to detect when previously retrieved data is incompatible with the device, the initial bytes written to `pData` **must** be a header consisting of the following members:

Table 5. Layout for pipeline cache header version `VK_PIPELINE_CACHE_HEADER_VERSION_ONE`

Offset	Size	Meaning
0	4	length in bytes of the entire pipeline cache header written as a stream of bytes, with the least significant byte first
4	4	a VkPipelineCacheHeaderVersion value written as a stream of bytes, with the least significant byte first
8	4	a vendor ID equal to `VkPhysicalDeviceProperties::vendorID` written as a stream of bytes, with the least significant byte first
12	4	a device ID equal to `VkPhysicalDeviceProperties::deviceID` written as a stream of bytes, with the least significant byte first
16	`VK_UUID_SIZE`	a pipeline cache ID equal to `VkPhysicalDeviceProperties::pipelineCacheUUID`

The first four bytes encode the length of the entire pipeline header, in bytes. This value includes all fields in the header including the pipeline cache version field and the size of the length field.

The next four bytes encode the pipeline cache version, as described for VkPipelineCacheHeaderVersion. A consumer of the pipeline cache **should** use the cache version to interpret the remainder of the cache header.

If `pDataSize` is less than what is necessary to store this header, nothing will be written to `pData` and zero will be written to `pDataSize`.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pipelineCache` **must** be a valid `VkPipelineCache` handle
- `pDataSize` **must** be a pointer to a `size_t` value
- If the value referenced by `pDataSize` is not `0`, and `pData` is not `NULL`, `pData` **must** be a pointer to an array of `pDataSize` bytes
- `pipelineCache` **must** have been created, allocated, or retrieved from `device`

Return Codes

Success

- `VK_SUCCESS`
- `VK_INCOMPLETE`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkDevice, VkPipelineCache

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetPipelineCacheData

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetQueryPoolResults(3)

Name

vkGetQueryPoolResults - Copy results of queries in a query pool to a host memory region

C Specification

To retrieve status and results for a set of queries, call:

```
VkResult vkGetQueryPoolResults(
    VkDevice                                    device,
    VkQueryPool                                 queryPool,
    uint32_t                                    firstQuery,
    uint32_t                                    queryCount,
    size_t                                      dataSize,
    void*                                       pData,
    VkDeviceSize                                stride,
    VkQueryResultFlags                          flags);
```

Parameters

- 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. firstQuery and queryCount together define a range of queries. For pipeline statistics queries, each query index in the pool contains one integer value for each bit that is enabled in VkQueryPoolCreateInfo::pipelineStatistics when the pool is created.
- dataSize is the size in bytes of the buffer pointed to by pData.
- pData is a pointer to a user-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.

Description

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. The behavior when not all queries are available, is described below.

If VK_QUERY_RESULT_64_BIT is not set and the result overflows a 32-bit value, the value **may** either wrap or saturate. Similarly, if VK_QUERY_RESULT_64_BIT is set and the result overflows a 64-bit value, the value **may** either wrap or saturate.

If VK_QUERY_RESULT_WAIT_BIT is set, Vulkan will wait for each query to be in the available state before retrieving the numerical results for that query. In this case, vkGetQueryPoolResults is guaranteed to succeed and return VK_SUCCESS if the queries become available in a finite time (i.e. if they have been issued and not reset). If queries will never finish (e.g. due to being reset but not issued), then vkGetQueryPoolResults **may** not return in finite time.

If VK_QUERY_RESULT_WAIT_BIT and VK_QUERY_RESULT_PARTIAL_BIT are both not set then no result values are written to pData for queries that are in the unavailable state at the time of the call, and vkGetQueryPoolResults returns VK_NOT_READY. However, availability state is still written to pData for those queries if VK_QUERY_RESULT_WITH_AVAILABILITY_BIT is set.

> *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 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 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 VK_QUERY_RESULT_PARTIAL_BIT is set, VK_QUERY_RESULT_WAIT_BIT is not set, and the query's status is unavailable, an intermediate result value between zero and the final result value is written to pData for that query.

VK_QUERY_RESULT_PARTIAL_BIT **must** not be used if the pool's queryType is VK_QUERY_TYPE_TIMESTAMP.

If VK_QUERY_RESULT_WITH_AVAILABILITY_BIT is set, the final integer value written for each query is non-zero if the query's status was available or zero if the status was unavailable. When VK_QUERY_RESULT_WITH_AVAILABILITY_BIT is used, implementations **must** guarantee that if they return a non-zero availability value then the numerical results **must** be valid, assuming the results are not reset by a subsequent command.

> *Note*
>
> Satisfying this guarantee **may** require careful ordering by the application, e.g. to read the availability status before reading the results.

Valid Usage

- firstQuery **must** be less than the number of queries in queryPool
- If VK_QUERY_RESULT_64_BIT is not set in flags then pData and stride **must** be multiples of 4
- If VK_QUERY_RESULT_64_BIT is set in flags then pData and stride **must** be multiples of 8
- The sum of firstQuery and queryCount **must** be less than or equal to the number of queries in queryPool
- dataSize **must** be large enough to contain the result of each query, as described here
- If the queryType used to create queryPool was VK_QUERY_TYPE_TIMESTAMP, flags **must** not contain VK_QUERY_RESULT_PARTIAL_BIT

Valid Usage (Implicit)

- device **must** be a valid VkDevice handle
- queryPool **must** be a valid VkQueryPool handle
- pData **must** be a pointer to an array of dataSize bytes
- flags **must** be a valid combination of VkQueryResultFlagBits values
- dataSize **must** be greater than 0
- queryPool **must** have been created, allocated, or retrieved from device

Return Codes

Success

- VK_SUCCESS
- VK_NOT_READY

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST

See Also

VkDevice, VkDeviceSize, VkQueryPool, VkQueryResultFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetQueryPoolResults

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkGetRenderAreaGranularity(3)

Name

vkGetRenderAreaGranularity - Returns the granularity for optimal render area

C Specification

To query the render area granularity, call:

```
void vkGetRenderAreaGranularity(
    VkDevice                                    device,
    VkRenderPass                                renderPass,
    VkExtent2D*                                 pGranularity);
```

Parameters

- `device` is the logical device that owns the render pass.
- `renderPass` is a handle to a render pass.
- `pGranularity` points to a VkExtent2D structure in which the granularity is returned.

Description

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` of the returned VkExtent2D (the vertical granularity).
- either the `offset.width` member in `renderArea` is a multiple of the horizontal granularity or `offset.x`+`offset.width` is equal to the `width` of the `framebuffer` in the VkRenderPassBeginInfo.
- either the `offset.height` member in `renderArea` is a multiple of the vertical granularity or `offset.y`+`offset.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)

- `device` **must** be a valid `VkDevice` handle
- `renderPass` **must** be a valid `VkRenderPass` handle
- `pGranularity` **must** be a pointer to a `VkExtent2D` structure
- `renderPass` **must** have been created, allocated, or retrieved from `device`

See Also

VkDevice, VkExtent2D, VkRenderPass

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkGetRenderAreaGranularity

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkInvalidateMappedMemoryRanges(3)

Name

vkInvalidateMappedMemoryRanges - Invalidate ranges of mapped memory objects

C Specification

To invalidate ranges of non-coherent memory from the host caches, call:

```
VkResult vkInvalidateMappedMemoryRanges(
    VkDevice                                    device,
    uint32_t                                    memoryRangeCount,
    const VkMappedMemoryRange*                  pMemoryRanges);
```

Parameters

- `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.

Description

`vkInvalidateMappedMemoryRanges` guarantees that device writes to the memory ranges described by `pMemoryRanges`, which have been made visible to 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.

> *Note*
>
> Mapping non-coherent memory does not implicitly invalidate the mapped memory, and device writes that have not been invalidated **must** be made visible before the host reads or overwrites them.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pMemoryRanges` **must** be a pointer to an array of `memoryRangeCount` valid `VkMappedMemoryRange` structures
- `memoryRangeCount` **must** be greater than `0`

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkDevice, VkMappedMemoryRange

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkInvalidateMappedMemoryRanges

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkMapMemory(3)

Name

vkMapMemory - Map a memory object into application address space

C Specification

To retrieve a host virtual address pointer to a region of a mappable memory object, call:

```
VkResult vkMapMemory(
    VkDevice                                    device,
    VkDeviceMemory                              memory,
    VkDeviceSize                                offset,
    VkDeviceSize                                size,
    VkMemoryMapFlags                            flags,
    void**                                      ppData);
```

Parameters

- `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` points to a pointer in which is returned a host-accessible pointer to the beginning of the mapped range. This pointer minus `offset` **must** be aligned to at least `VkPhysicalDeviceLimits` ::`minMemoryMapAlignment`.

Description

It is an application error to call `vkMapMemory` on a memory object that is already 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 mapped for host access, 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.

Valid Usage

- `memory` **must** not currently be mapped
- `offset` **must** be less than the size of `memory`
- If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be greater than `0`
- If `size` is not equal to `VK_WHOLE_SIZE`, `size` **must** be less than or equal to the size of the `memory` minus `offset`
- `memory` **must** have been created with a memory type that reports `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT`

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `memory` **must** be a valid `VkDeviceMemory` handle
- `flags` **must** be `0`
- `ppData` **must** be a pointer to a pointer
- `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_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_MEMORY_MAP_FAILED

See Also

VkDevice, VkDeviceMemory, VkDeviceSize, VkMemoryMapFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkMapMemory

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkMergePipelineCaches(3)

Name

vkMergePipelineCaches - Combine the data stores of pipeline caches

C Specification

Pipeline cache objects **can** be merged using the command:

```
VkResult vkMergePipelineCaches(
    VkDevice                                    device,
    VkPipelineCache                             dstCache,
    uint32_t                                    srcCacheCount,
    const VkPipelineCache*                      pSrcCaches);
```

Parameters

- `device` is the logical device that owns the pipeline cache objects.
- `dstCache` is the handle of the pipeline cache to merge results into.
- `srcCacheCount` is the length of the `pSrcCaches` array.
- `pSrcCaches` is an array of pipeline cache handles, which will be merged into `dstCache`. The previous contents of `dstCache` are included after the merge.

Description

Note

The details of the merge operation are implementation dependent, but implementations **should** merge the contents of the specified pipelines and prune duplicate entries.

Valid Usage

- `dstCache` **must** not appear in the list of source caches

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `dstCache` **must** be a valid `VkPipelineCache` handle
- `pSrcCaches` **must** be a pointer to an array of `srcCacheCount` valid `VkPipelineCache` handles
- `srcCacheCount` **must** be greater than `0`
- `dstCache` **must** have been created, allocated, or retrieved from `device`
- Each element of `pSrcCaches` **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `dstCache` **must** be externally synchronized

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkDevice, VkPipelineCache

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkMergePipelineCaches

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkQueueBindSparse(3)

Name

vkQueueBindSparse - Bind device memory to a sparse resource object

C Specification

To submit sparse binding operations to a queue, call:

```
VkResult vkQueueBindSparse(
    VkQueue                                     queue,
    uint32_t                                    bindInfoCount,
    const VkBindSparseInfo*                     pBindInfo,
    VkFence                                     fence);
```

Parameters

- `queue` is the queue that the sparse binding operations will be submitted to.
- `bindInfoCount` is the number of elements in the `pBindInfo` array.
- `pBindInfo` is an array of VkBindSparseInfo structures, each specifying a sparse binding submission batch.
- `fence` is an optional handle to a fence to be signaled. If `fence` is not VK_NULL_HANDLE, it defines a fence signal operation.

Description

`vkQueueBindSparse` is a queue submission command, with each batch defined by an element of `pBindInfo` as an instance of the VkBindSparseInfo structure. Batches begin execution in the order they appear in `pBindInfo`, but **may** complete out of order.

Within a batch, a given range of a resource **must** not be bound more than once. Across batches, if a range is to be bound to one allocation and offset and then to another allocation and offset, then the application **must** guarantee (usually using semaphores) that the binding operations are executed in the correct order, as well as to order binding operations against the execution of command buffer submissions.

As no operation to vkQueueBindSparse causes any pipeline stage to access memory, synchronization primitives used in this command effectively only define execution dependencies.

Additional information about fence and semaphore operation is described in the synchronization chapter.

Valid Usage

- If `fence` is not VK_NULL_HANDLE, `fence` **must** be unsignaled
- 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
- Any given element of the `pSignalSemaphores` member of any element of `pBindInfo` **must** be unsignaled when the semaphore signal operation it defines is executed on the device
- When a semaphore unsignal operation defined by any element of the `pWaitSemaphores` member of any element of `pBindInfo` executes on `queue`, no other queue **must** be waiting on the same semaphore.
- All elements of the `pWaitSemaphores` member of all elements of `pBindInfo` **must** be semaphores that are signaled, or have semaphore signal operations previously submitted for execution.

Valid Usage (Implicit)

- `queue` **must** be a valid `VkQueue` handle
- If `bindInfoCount` is not `0`, `pBindInfo` **must** be a pointer to an array of `bindInfoCount` valid `VkBindSparseInfo` structures
- If `fence` is not VK_NULL_HANDLE, `fence` **must** be a valid `VkFence` handle
- The `queue` **must** support sparse binding operations
- Both of `fence`, and `queue` that are valid handles **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `queue` **must** be externally synchronized
- Host access to `pBindInfo`[].pWaitSemaphores[] **must** be externally synchronized
- Host access to `pBindInfo`[].pSignalSemaphores[] **must** be externally synchronized
- Host access to `pBindInfo`[].pBufferBinds[].buffer **must** be externally synchronized
- Host access to `pBindInfo`[].pImageOpaqueBinds[].image **must** be externally synchronized
- Host access to `pBindInfo`[].pImageBinds[].image **must** be externally synchronized
- Host access to `fence` **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Pipeline Type
-	-	SPARSE_BINDING	-

Return Codes

Success

- VK_SUCCESS

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY
- VK_ERROR_DEVICE_LOST

See Also

VkBindSparseInfo, VkFence, VkQueue

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkQueueBindSparse

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkQueueSubmit(3)

Name

vkQueueSubmit - Submits a sequence of semaphores or command buffers to a queue

C Specification

To submit command buffers to a queue, call:

```
VkResult vkQueueSubmit(
    VkQueue                                     queue,
    uint32_t                                    submitCount,
    const VkSubmitInfo*                         pSubmits,
    VkFence                                     fence);
```

Parameters

- `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.
- `fence` is an optional handle to a fence to be signaled. If `fence` is not VK_NULL_HANDLE, it defines a fence signal operation.

Description

> *Note*
>
> Submission can be a high overhead operation, and applications **should** attempt to batch work together into as few calls to `vkQueueSubmit` as possible.

`vkQueueSubmit` is a queue submission command, with each batch defined by an element of `pSubmits` as an instance of the VkSubmitInfo structure. 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 back 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 **can** not make that guarantee, the implementation **must** return `VK_ERROR_DEVICE_LOST`. See Lost Device.

Valid Usage

- If `fence` is not VK_NULL_HANDLE, `fence` **must** be unsignaled
- 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
- 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*.
- 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.
- Any given 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
- When a semaphore unsignal operation defined by any element of the `pWaitSemaphores` member of any element of `pSubmits` executes on `queue`, no other queue **must** be waiting on the same semaphore.
- All elements of the `pWaitSemaphores` member of all elements of `pSubmits` **must** be semaphores that are signaled, or have semaphore signal operations previously submitted for execution.
- Any given element of the `pCommandBuffers` member of any element of `pSubmits` **must** be in the pending or executable state.
- If any given 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.
- Any secondary command buffers recorded into any given element of the `pCommandBuffers` member of any element of `pSubmits` **must** be in the pending or executable state.
- If any secondary command buffers recorded into any given 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.
- Any given element of the `pCommandBuffers` member of any element of `pSubmits` **must** have been allocated from a `VkCommandPool` that was created for the same queue family `queue` belongs to.

Valid Usage (Implicit)

- `queue` **must** be a valid `VkQueue` handle
- If `submitCount` is not `0`, `pSubmits` **must** be a pointer to an array of `submitCount` valid `VkSubmitInfo` structures
- If `fence` is not VK_NULL_HANDLE, `fence` **must** be a valid `VkFence` handle
- Both of `fence`, and `queue` that are valid handles **must** have been created, allocated, or retrieved from the same `VkDevice`

Host Synchronization

- Host access to `queue` **must** be externally synchronized
- Host access to `pSubmits`[].pWaitSemaphores[] **must** be externally synchronized
- Host access to `pSubmits`[].pSignalSemaphores[] **must** be externally synchronized
- Host access to `fence` **must** be externally synchronized

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Pipeline Type
-	-	Any	-

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`

See Also

VkFence, VkQueue, VkSubmitInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkQueueSubmit

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkQueueWaitIdle(3)

Name

vkQueueWaitIdle - Wait for a queue to become idle

C Specification

To wait on the host for the completion of outstanding queue operations for a given queue, call:

```
VkResult vkQueueWaitIdle(
    VkQueue                                     queue);
```

Parameters

- `queue` is the queue on which to wait.

Description

`vkQueueWaitIdle` is equivalent to submitting a fence to a queue and waiting with an infinite timeout for that fence to signal.

Valid Usage (Implicit)

- `queue` **must** be a valid `VkQueue` handle

Command Properties

Command Buffer Levels	Render Pass Scope	Supported Queue Types	Pipeline Type
-	-	Any	-

Return Codes

Success

- `VK_SUCCESS`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`

See Also

VkQueue

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkQueueWaitIdle

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkResetCommandBuffer(3)

Name

vkResetCommandBuffer - Reset a command buffer to the initial state

C Specification

To reset command buffers, call:

```
VkResult vkResetCommandBuffer(
    VkCommandBuffer                             commandBuffer,
    VkCommandBufferResetFlags                   flags);
```

Parameters

- `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.

Description

Any primary command buffer that is in the recording or executable state and has `commandBuffer` recorded into it, becomes invalid.

Valid Usage

- `commandBuffer` **must** not be in the pending state
- `commandBuffer` **must** have been allocated from a pool that was created with the `VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT`

Valid Usage (Implicit)

- `commandBuffer` **must** be a valid `VkCommandBuffer` handle
- `flags` **must** be a valid combination of VkCommandBufferResetFlagBits values

Host Synchronization

- Host access to `commandBuffer` **must** be externally synchronized

Return Codes

Success

- VK_SUCCESS

Failure

- VK_ERROR_OUT_OF_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

See Also

VkCommandBuffer, VkCommandBufferResetFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkResetCommandBuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkResetCommandPool(3)

Name

vkResetCommandPool - Reset a command pool

C Specification

To reset a command pool, call:

```
VkResult vkResetCommandPool(
    VkDevice                                    device,
    VkCommandPool                               commandPool,
    VkCommandPoolResetFlags                     flags);
```

Parameters

- `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.

Description

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

- All `VkCommandBuffer` objects allocated from `commandPool` **must** not be in the pending state

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `commandPool` **must** be a valid `VkCommandPool` handle
- `flags` **must** be a valid combination of VkCommandPoolResetFlagBits values
- `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_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkCommandPool, VkCommandPoolResetFlags, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkResetCommandPool

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkResetDescriptorPool(3)

Name

vkResetDescriptorPool - Resets a descriptor pool object

C Specification

To return all descriptor sets allocated from a given pool to the pool, rather than freeing individual descriptor sets, call:

```
VkResult vkResetDescriptorPool(
    VkDevice                                    device,
    VkDescriptorPool                            descriptorPool,
    VkDescriptorPoolResetFlags                  flags);
```

Parameters

- `device` is the logical device that owns the descriptor pool.
- `descriptorPool` is the descriptor pool to be reset.
- `flags` is reserved for future use.

Description

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

- All uses of `descriptorPool` (via any allocated descriptor sets) **must** have completed execution

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `descriptorPool` **must** be a valid `VkDescriptorPool` handle
- `flags` **must** be `0`
- `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_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkDescriptorPool, VkDescriptorPoolResetFlags, VkDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkResetDescriptorPool

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkResetEvent(3)

Name

vkResetEvent - Reset an event to non-signaled state

C Specification

To set the state of an event to unsignaled from the host, call:

```
VkResult vkResetEvent(
    VkDevice                                    device,
    VkEvent                                     event);
```

Parameters

- `device` is the logical device that owns the event.
- `event` is the event to reset.

Description

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

- `event` **must** not be waited on by a `vkCmdWaitEvents` command that is currently executing

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `event` **must** be a valid `VkEvent` handle
- `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_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

See Also

VkDevice, VkEvent

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkResetEvent

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkResetFences(3)

Name

vkResetFences - Resets one or more fence objects

C Specification

To set the state of fences to unsignaled from the host, call:

```
VkResult vkResetFences(
    VkDevice                                    device,
    uint32_t                                    fenceCount,
    const VkFence*                              pFences);
```

Parameters

- `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.

Description

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

- Any given element of `pFences` **must** not currently be associated with any queue command that has not yet completed execution on that queue

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pFences` **must** be a pointer to an array of `fenceCount` valid `VkFence` handles
- `fenceCount` **must** be greater than `0`
- 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_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`

See Also

VkDevice, VkFence

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkResetFences

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkSetEvent(3)

Name

vkSetEvent - Set an event to signaled state

C Specification

To set the state of an event to signaled from the host, call:

```
VkResult vkSetEvent(
    VkDevice                                    device,
    VkEvent                                     event);
```

Parameters

- `device` is the logical device that owns the event.
- `event` is the event to set.

Description

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.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `event` **must** be a valid `VkEvent` handle
- `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_HOST_MEMORY
- VK_ERROR_OUT_OF_DEVICE_MEMORY

See Also

VkDevice, VkEvent

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkSetEvent

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkUnmapMemory(3)

Name

vkUnmapMemory - Unmap a previously mapped memory object

C Specification

To unmap a memory object once host access to it is no longer needed by the application, call:

```
void vkUnmapMemory(
    VkDevice                                    device,
    VkDeviceMemory                              memory);
```

Parameters

- `device` is the logical device that owns the memory.
- `memory` is the memory object to be unmapped.

Description

Valid Usage

- `memory` **must** currently be mapped

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `memory` **must** be a valid `VkDeviceMemory` handle
- `memory` **must** have been created, allocated, or retrieved from `device`

Host Synchronization

- Host access to `memory` **must** be externally synchronized

See Also

VkDevice, VkDeviceMemory

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkUnmapMemory

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkUpdateDescriptorSets(3)

Name

vkUpdateDescriptorSets - Update the contents of a descriptor set object

C Specification

Once allocated, descriptor sets **can** be updated with a combination of write and copy operations. To update descriptor sets, call:

```
void vkUpdateDescriptorSets(
    VkDevice                                    device,
    uint32_t                                    descriptorWriteCount,
    const VkWriteDescriptorSet*                 pDescriptorWrites,
    uint32_t                                    descriptorCopyCount,
    const VkCopyDescriptorSet*                  pDescriptorCopies);
```

Parameters

- `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.

Description

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 given 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, that command buffer becomes invalid.

Valid Usage

- The `dstSet` member of any given element of `pDescriptorWrites` or `pDescriptorCopies` **must** not be used by any command that was recorded to a command buffer which is in the pending state.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- If `descriptorWriteCount` is not `0`, `pDescriptorWrites` **must** be a pointer to an array of `descriptorWriteCount` valid `VkWriteDescriptorSet` structures
- If `descriptorCopyCount` is not `0`, `pDescriptorCopies` **must** be a pointer to an array of `descriptorCopyCount` valid `VkCopyDescriptorSet` structures

Host Synchronization

- Host access to `pDescriptorWrites`[].dstSet **must** be externally synchronized
- Host access to `pDescriptorCopies`[].dstSet **must** be externally synchronized

See Also

VkCopyDescriptorSet, VkDevice, VkWriteDescriptorSet

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkUpdateDescriptorSets

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

vkWaitForFences(3)

Name

vkWaitForFences - Wait for one or more fences to become signaled

C Specification

To wait for one or more fences to enter the signaled state on the host, call:

```
VkResult vkWaitForFences(
    VkDevice                                    device,
    uint32_t                                    fenceCount,
    const VkFence*                              pFences,
    VkBool32                                    waitAll,
    uint64_t                                    timeout);
```

Parameters

- `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.

Description

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 up to `timeout` nanoseconds for the condition to become satisfied.

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 specified timeout period expires before the condition is satisfied, `vkWaitForFences` returns `VK_TIMEOUT`. If the condition is satisfied before `timeout` nanoseconds has expired, `vkWaitForFences` returns `VK_SUCCESS`.

If device loss occurs (see Lost Device) before the timeout has expired, `vkWaitForFences` **must** return in finite time with either `VK_SUCCESS` or `VK_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 client 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.

Valid Usage (Implicit)

- `device` **must** be a valid `VkDevice` handle
- `pFences` **must** be a pointer to an array of `fenceCount` valid `VkFence` handles
- `fenceCount` **must** be greater than `0`
- Each element of `pFences` **must** have been created, allocated, or retrieved from `device`

Return Codes

Success

- `VK_SUCCESS`
- `VK_TIMEOUT`

Failure

- `VK_ERROR_OUT_OF_HOST_MEMORY`
- `VK_ERROR_OUT_OF_DEVICE_MEMORY`
- `VK_ERROR_DEVICE_LOST`

See Also

`VkBool32`, VkDevice, VkFence

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#vkWaitForFences

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

Object Handles

VkBuffer(3)

Name

VkBuffer - Opaque handle to a buffer object

C Specification

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 via certain commands, or by directly specifying them as parameters to certain commands.

Buffers are represented by `VkBuffer` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkBuffer)
```

Description

See Also

VkBufferMemoryBarrier, VkBufferViewCreateInfo, VkDescriptorBufferInfo, VkSparseBufferMemoryBindInfo, vkBindBufferMemory, vkCmdBindIndexBuffer, vkCmdBindVertexBuffers, vkCmdCopyBuffer, vkCmdCopyBufferToImage, vkCmdCopyImageToBuffer, vkCmdCopyQueryPoolResults, vkCmdDispatchIndirect, vkCmdDrawIndexedIndirect, vkCmdDrawIndirect, vkCmdFillBuffer, vkCmdUpdateBuffer, vkCreateBuffer, vkDestroyBuffer, vkGetBufferMemoryRequirements

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkBufferView(3)

Name

VkBufferView - Opaque handle to a buffer view object

C Specification

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 interpreted as formatted data. 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:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkBufferView)
```

Description

See Also

VkWriteDescriptorSet, vkCreateBufferView, vkDestroyBufferView

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBufferView

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCommandBuffer(3)

Name

VkCommandBuffer - Opaque handle to a command buffer object

C Specification

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:

```
VK_DEFINE_HANDLE(VkCommandBuffer)
```

Description

See Also

VkSubmitInfo, vkAllocateCommandBuffers, vkBeginCommandBuffer, 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, vkEndCommandBuffer, vkFreeCommandBuffers, vkResetCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandBuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCommandPool(3)

Name

VkCommandPool - Opaque handle to a command pool object

C Specification

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 are represented by `VkCommandPool` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkCommandPool)
```

Description

See Also

VkCommandBufferAllocateInfo, vkCreateCommandPool, vkDestroyCommandPool, vkFreeCommandBuffers, vkResetCommandPool

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandPool

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorPool(3)

Name

VkDescriptorPool - Opaque handle to a descriptor pool object

C Specification

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 are represented by `VkDescriptorPool` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDescriptorPool)
```

Description

See Also

VkDescriptorSetAllocateInfo, vkCreateDescriptorPool, vkDestroyDescriptorPool, vkFreeDescriptorSets, vkResetDescriptorPool

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorPool

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorSet(3)

Name

VkDescriptorSet - Opaque handle to a descriptor set object

C Specification

Descriptor sets are allocated from descriptor pool objects, and are represented by `VkDescriptorSet` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDescriptorSet)
```

Description

See Also

VkCopyDescriptorSet, VkWriteDescriptorSet, vkAllocateDescriptorSets, vkCmdBindDescriptorSets, vkFreeDescriptorSets

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorSet

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorSetLayout(3)

Name

VkDescriptorSetLayout - Opaque handle to a descriptor set layout object

C Specification

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:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDescriptorSetLayout)
```

Description

See Also

VkDescriptorSetAllocateInfo, VkPipelineLayoutCreateInfo, vkCreateDescriptorSetLayout, vkDestroyDescriptorSetLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorSetLayout

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDevice(3)

Name

VkDevice - Opaque handle to a device object

C Specification

Logical devices are represented by `VkDevice` handles:

```
VK_DEFINE_HANDLE(VkDevice)
```

Description

See Also

vkAllocateCommandBuffers, vkAllocateDescriptorSets, vkAllocateMemory, vkBindBufferMemory, vkBindImageMemory, vkCreateBuffer, vkCreateBufferView, vkCreateCommandPool, vkCreateComputePipelines, vkCreateDescriptorPool, vkCreateDescriptorSetLayout, vkCreateDevice, vkCreateEvent, vkCreateFence, vkCreateFramebuffer, vkCreateGraphicsPipelines, vkCreateImage, vkCreateImageView, vkCreatePipelineCache, vkCreatePipelineLayout, vkCreateQueryPool, vkCreateRenderPass, vkCreateSampler, vkCreateSemaphore, vkCreateShaderModule, vkDestroyBuffer, vkDestroyBufferView, vkDestroyCommandPool, vkDestroyDescriptorPool, vkDestroyDescriptorSetLayout, vkDestroyDevice, vkDestroyEvent, vkDestroyFence, vkDestroyFramebuffer, vkDestroyImage, vkDestroyImageView, vkDestroyPipeline, vkDestroyPipelineCache, vkDestroyPipelineLayout, vkDestroyQueryPool, vkDestroyRenderPass, vkDestroySampler, vkDestroySemaphore, vkDestroyShaderModule, vkDeviceWaitIdle, vkFlushMappedMemoryRanges, vkFreeCommandBuffers, vkFreeDescriptorSets, vkFreeMemory, vkGetBufferMemoryRequirements, vkGetDeviceMemoryCommitment, vkGetDeviceProcAddr, vkGetDeviceQueue, vkGetEventStatus, vkGetFenceStatus, vkGetImageMemoryRequirements, vkGetImageSparseMemoryRequirements, vkGetImageSubresourceLayout, vkGetPipelineCacheData, vkGetQueryPoolResults, vkGetRenderAreaGranularity, vkInvalidateMappedMemoryRanges, vkMapMemory, vkMergePipelineCaches, vkResetCommandPool, vkResetDescriptorPool, vkResetEvent, vkResetFences, vkSetEvent, vkUnmapMemory, vkUpdateDescriptorSets, vkWaitForFences

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDevice

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDeviceMemory(3)

Name

VkDeviceMemory - Opaque handle to a device memory object

C Specification

A Vulkan device operates on data in device memory via memory objects that are represented in the API by a `VkDeviceMemory` handle.

Memory objects are represented by `VkDeviceMemory` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkDeviceMemory)
```

Description

See Also

VkMappedMemoryRange, VkSparseImageMemoryBind, VkSparseMemoryBind, vkAllocateMemory, vkBindBufferMemory, vkBindImageMemory, vkFreeMemory, vkGetDeviceMemoryCommitment, vkMapMemory, vkUnmapMemory

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDeviceMemory

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkEvent(3)

Name

VkEvent - Opaque handle to a event object

C Specification

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 have two states - signaled and unsignaled. An application **can** signal an event, or unsignal it, on either the host or the device. A device **can** 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:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkEvent)
```

Description

See Also

vkCmdResetEvent, vkCmdSetEvent, vkCmdWaitEvents, vkCreateEvent, vkDestroyEvent, vkGetEventStatus, vkResetEvent, vkSetEvent

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkEvent

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkFence(3)

Name

VkFence - Opaque handle to a fence object

C Specification

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.

Fences are represented by `VkFence` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkFence)
```

Description

See Also

vkCreateFence, vkDestroyFence, vkGetFenceStatus, vkQueueBindSparse, vkQueueSubmit, vkResetFences, vkWaitForFences

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFence

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkFramebuffer(3)

Name

VkFramebuffer - Opaque handle to a framebuffer object

C Specification

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:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkFramebuffer)
```

Description

See Also

VkCommandBufferInheritanceInfo, VkRenderPassBeginInfo, vkCreateFramebuffer, vkDestroyFramebuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFramebuffer

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImage(3)

Name

VkImage - Opaque handle to a image object

C Specification

Images represent multidimensional - up to 3 - arrays of data which **can** be used for various purposes (e.g. attachments, textures), by binding them to a graphics or compute pipeline via descriptor sets, or by directly specifying them as parameters to certain commands.

Images are represented by `VkImage` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkImage)
```

Description

See Also

VkImageMemoryBarrier, VkImageViewCreateInfo, VkSparseImageMemoryBindInfo, VkSparseImageOpaqueMemoryBindInfo, vkBindImageMemory, vkCmdBlitImage, vkCmdClearColorImage, vkCmdClearDepthStencilImage, vkCmdCopyBufferToImage, vkCmdCopyImage, vkCmdCopyImageToBuffer, vkCmdResolveImage, vkCreateImage, vkDestroyImage, vkGetImageMemoryRequirements, vkGetImageSparseMemoryRequirements, vkGetImageSubresourceLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImage

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageView(3)

Name

VkImageView - Opaque handle to a image view object

C Specification

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:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkImageView)
```

Description

See Also

VkDescriptorImageInfo, VkFramebufferCreateInfo, vkCreateImageView, vkDestroyImageView

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageView

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkInstance(3)

Name

VkInstance - Opaque handle to a instance object

C Specification

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:

```
VK_DEFINE_HANDLE(VkInstance)
```

Description

See Also

vkCreateInstance, vkDestroyInstance, vkEnumeratePhysicalDevices, vkGetInstanceProcAddr

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkInstance

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPhysicalDevice(3)

Name

VkPhysicalDevice - Opaque handle to a physical device object

C Specification

Vulkan separates the concept of *physical* and *logical* devices. A physical device usually represents a single device in a system (perhaps made up of several individual hardware devices working together), of which there are a finite number. A logical device represents an application's view of the device.

Physical devices are represented by `VkPhysicalDevice` handles:

```
VK_DEFINE_HANDLE(VkPhysicalDevice)
```

Description

See Also

vkCreateDevice, vkEnumerateDeviceExtensionProperties, vkEnumerateDeviceLayerProperties, vkEnumeratePhysicalDevices, vkGetPhysicalDeviceFeatures, vkGetPhysicalDeviceFormatProperties, vkGetPhysicalDeviceImageFormatProperties, vkGetPhysicalDeviceMemoryProperties, vkGetPhysicalDeviceProperties, vkGetPhysicalDeviceQueueFamilyProperties, vkGetPhysicalDeviceSparseImageFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPhysicalDevice

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipeline(3)

Name

VkPipeline - Opaque handle to a pipeline object

C Specification

Compute and graphics pipelines are each represented by `VkPipeline` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkPipeline)
```

Description

See Also

VkComputePipelineCreateInfo, VkGraphicsPipelineCreateInfo, vkCmdBindPipeline, vkCreateComputePipelines, vkCreateGraphicsPipelines, vkDestroyPipeline

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipeline

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineCache(3)

Name

VkPipelineCache - Opaque handle to a pipeline cache object

C Specification

Pipeline cache objects allow the result of pipeline construction to be reused between pipelines and between runs of an application. Reuse between pipelines is achieved by passing the same pipeline cache object when creating multiple related pipelines. Reuse across runs of an application is achieved by retrieving pipeline cache contents in one run of an application, saving the contents, and using them to preinitialize a pipeline cache on a subsequent run. The contents of the pipeline cache objects are managed by the implementation. Applications **can** manage the host memory consumed by a pipeline cache object and control the amount of data retrieved from a pipeline cache object.

Pipeline cache objects are represented by `VkPipelineCache` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkPipelineCache)
```

Description

See Also

vkCreateComputePipelines, vkCreateGraphicsPipelines, vkCreatePipelineCache, vkDestroyPipelineCache, vkGetPipelineCacheData, vkMergePipelineCaches

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineCache

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineLayout(3)

Name

VkPipelineLayout - Opaque handle to a pipeline layout object

C Specification

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 which describes 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:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkPipelineLayout)
```

Description

See Also

VkComputePipelineCreateInfo, VkGraphicsPipelineCreateInfo, vkCmdBindDescriptorSets, vkCmdPushConstants, vkCreatePipelineLayout, vkDestroyPipelineLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineLayout

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkQueryPool(3)

Name

VkQueryPool - Opaque handle to a query pool object

C Specification

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 are represented by `VkQueryPool` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkQueryPool)
```

Description

See Also

vkCmdBeginQuery, vkCmdCopyQueryPoolResults, vkCmdEndQuery, vkCmdResetQueryPool, vkCmdWriteTimestamp, vkCreateQueryPool, vkDestroyQueryPool, vkGetQueryPoolResults

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueryPool

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkQueue(3)

Name

VkQueue - Opaque handle to a queue object

C Specification

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 are represented by `VkQueue` handles:

```
VK_DEFINE_HANDLE(VkQueue)
```

Description

See Also

vkGetDeviceQueue, vkQueueBindSparse, vkQueueSubmit, vkQueueWaitIdle

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueue

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkRenderPass(3)

Name

VkRenderPass - Opaque handle to a render pass object

C Specification

A *render pass* represents a collection of attachments, subpasses, and dependencies between the subpasses, and describes how the attachments are used over the course of the subpasses. The use of a render pass in a command buffer is a *render pass instance*.

Render passes are represented by `VkRenderPass` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkRenderPass)
```

Description

See Also

VkCommandBufferInheritanceInfo, VkFramebufferCreateInfo, VkGraphicsPipelineCreateInfo, VkRenderPassBeginInfo, vkCreateRenderPass, vkDestroyRenderPass, vkGetRenderAreaGranularity

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkRenderPass

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSampler(3)

Name

VkSampler - Opaque handle to a sampler object

C Specification

`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:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSampler)
```

Description

See Also

VkDescriptorImageInfo, VkDescriptorSetLayoutBinding, vkCreateSampler, vkDestroySampler

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSampler

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSemaphore(3)

Name

VkSemaphore - Opaque handle to a semaphore object

C Specification

Semaphores are a synchronization primitive that **can** be used to insert a dependency between batches submitted to queues. Semaphores have two states - signaled and unsignaled. The state of a semaphore **can** be signaled after execution of a batch of commands is completed. A batch **can** wait for a semaphore to become signaled before it begins execution, and the semaphore is also unsignaled before the batch begins execution.

Semaphores are represented by `VkSemaphore` handles:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkSemaphore)
```

Description

See Also

VkBindSparseInfo, VkSubmitInfo, vkCreateSemaphore, vkDestroySemaphore

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSemaphore

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkShaderModule(3)

Name

VkShaderModule - Opaque handle to a shader module object

C Specification

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:

```
VK_DEFINE_NON_DISPATCHABLE_HANDLE(VkShaderModule)
```

Description

See Also

VkPipelineShaderStageCreateInfo, vkCreateShaderModule, vkDestroyShaderModule

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkShaderModule

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

Structures

VkAllocationCallbacks(3)

Name

VkAllocationCallbacks - Structure containing callback function pointers for memory allocation

C Specification

Allocators are provided by the application as a pointer to a `VkAllocationCallbacks` structure:

```
typedef struct VkAllocationCallbacks {
    void*                                   pUserData;
    PFN_vkAllocationFunction                pfnAllocation;
    PFN_vkReallocationFunction              pfnReallocation;
    PFN_vkFreeFunction                      pfnFree;
    PFN_vkInternalAllocationNotification    pfnInternalAllocation;
    PFN_vkInternalFreeNotification          pfnInternalFree;
} VkAllocationCallbacks;
```

Members

- `pUserData` is a value to be interpreted by the implementation of the callbacks. When any of the callbacks in `VkAllocationCallbacks` are called, the Vulkan implementation will pass this value as the first parameter to the callback. This value **can** vary each time an allocator is passed into a command, even when the same object takes an allocator in multiple commands.
- `pfnAllocation` is a pointer to an application-defined memory allocation function of type PFN_vkAllocationFunction.
- `pfnReallocation` is a pointer to an application-defined memory reallocation function of type PFN_vkReallocationFunction.
- `pfnFree` is a pointer to an application-defined memory free function of type PFN_vkFreeFunction.
- `pfnInternalAllocation` is a pointer to an application-defined function that is called by the implementation when the implementation makes internal allocations, and it is of type PFN_vkInternalAllocationNotification.
- `pfnInternalFree` is a pointer to an application-defined function that is called by the implementation when the implementation frees internal allocations, and it is of type PFN_vkInternalFreeNotification.

Description

Valid Usage

- `pfnAllocation` **must** be a pointer to a valid user-defined PFN_vkAllocationFunction
- `pfnReallocation` **must** be a pointer to a valid user-defined PFN_vkReallocationFunction
- `pfnFree` **must** be a pointer to a valid user-defined PFN_vkFreeFunction
- If either of `pfnInternalAllocation` or `pfnInternalFree` is not `NULL`, both **must** be valid callbacks

See Also

PFN_vkAllocationFunction, PFN_vkFreeFunction, PFN_vkInternalAllocationNotification, PFN_vkInternalFreeNotification, PFN_vkReallocationFunction, vkAllocateMemory, 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, vkFreeMemory

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkAllocationCallbacks

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkApplicationInfo(3)

Name

VkApplicationInfo - Structure specifying application info

C Specification

The `VkApplicationInfo` structure is defined as:

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

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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` is the version of the Vulkan API against which the application expects to run, encoded as described in the API Version Numbers and Semantics section. If `apiVersion` is 0 the implementation **must** ignore it, otherwise if the implementation does not support the requested `apiVersion`, or an effective substitute for `apiVersion`, it **must** return `VK_ERROR_INCOMPATIBLE_DRIVER`. The patch version number specified in `apiVersion` is ignored when creating an instance object. Only the major and minor versions of the instance **must** match those requested in `apiVersion`.

Description

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_APPLICATION_INFO`
- `pNext` **must** be `NULL`
- If `pApplicationName` is not `NULL`, `pApplicationName` **must** be a null-terminated UTF-8 string
- If `pEngineName` is not `NULL`, `pEngineName` **must** be a null-terminated UTF-8 string

See Also

VkInstanceCreateInfo, VkStructureType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkApplicationInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkAttachmentDescription(3)

Name

VkAttachmentDescription - Structure specifying an attachment description

C Specification

The `VkAttachmentDescription` structure is defined as:

```
typedef struct VkAttachmentDescription {
    VkAttachmentDescriptionFlags    flags;
    VkFormat                        format;
    VkSampleCountFlagBits           samples;
    VkAttachmentLoadOp              loadOp;
    VkAttachmentStoreOp             storeOp;
    VkAttachmentLoadOp              stencilLoadOp;
    VkAttachmentStoreOp             stencilStoreOp;
    VkImageLayout                   initialLayout;
    VkImageLayout                   finalLayout;
} VkAttachmentDescription;
```

Members

- `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 the number of samples of the image as defined in VkSampleCountFlagBits.
- `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. During a render pass instance, an attachment **can** use a different layout in each subpass, if desired.

Description

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* that execute as part of the first subpass that uses the attachment. storeOp and stencilStoreOp define the *store operations* that execute as part of the last subpass that uses the attachment.

The load operation for each value in an attachment used by a subpass happens-before any command recorded into that subpass reads from that value. 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.

Store operations for each value in an attachment used by a subpass happen-after any command recorded into that subpass writes to that value. 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.

If an attachment is not used by any subpass, then loadOp, storeOp, stencilStoreOp, and stencilLoadOp are ignored, and the attachment's memory contents will not be modified by execution of a render pass instance.

During a render pass instance, input/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.

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.

Valid Usage

- finalLayout **must** not be VK_IMAGE_LAYOUT_UNDEFINED or VK_IMAGE_LAYOUT_PREINITIALIZED

Valid Usage (Implicit)

- `flags` **must** be a valid combination of VkAttachmentDescriptionFlagBits values
- `format` **must** be a valid VkFormat value
- `samples` **must** be a valid VkSampleCountFlagBits value
- `loadOp` **must** be a valid VkAttachmentLoadOp value
- `storeOp` **must** be a valid VkAttachmentStoreOp value
- `stencilLoadOp` **must** be a valid VkAttachmentLoadOp value
- `stencilStoreOp` **must** be a valid VkAttachmentStoreOp value
- `initialLayout` **must** be a valid VkImageLayout value
- `finalLayout` **must** be a valid VkImageLayout value

See Also

VkAttachmentDescriptionFlags, VkAttachmentLoadOp, VkAttachmentStoreOp, VkFormat, VkImageLayout, VkRenderPassCreateInfo, VkSampleCountFlagBits

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkAttachmentDescription

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkAttachmentReference(3)

Name

VkAttachmentReference - Structure specifying an attachment reference

C Specification

The `VkAttachmentReference` structure is defined as:

```
typedef struct VkAttachmentReference {
    uint32_t         attachment;
    VkImageLayout    layout;
} VkAttachmentReference;
```

Members

- `attachment` is the index of the attachment of the render pass, and corresponds to the index of the corresponding element in the `pAttachments` array of the `VkRenderPassCreateInfo` structure. If any color or depth/stencil attachments are `VK_ATTACHMENT_UNUSED`, then no writes occur for those attachments.
- `layout` is a VkImageLayout value specifying the layout the attachment uses during the subpass.

Description

Valid Usage

- `layout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`

Valid Usage (Implicit)

- `layout` **must** be a valid VkImageLayout value

See Also

VkImageLayout, VkSubpassDescription

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkAttachmentReference

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkBindSparseInfo(3)

Name

VkBindSparseInfo - Structure specifying a sparse binding operation

C Specification

The `VkBindSparseInfo` structure is defined as:

```
typedef struct VkBindSparseInfo {
    VkStructureType                             sType;
    const void*                                 pNext;
    uint32_t                                    waitSemaphoreCount;
    const VkSemaphore*                          pWaitSemaphores;
    uint32_t                                    bufferBindCount;
    const VkSparseBufferMemoryBindInfo*         pBufferBinds;
    uint32_t                                    imageOpaqueBindCount;
    const VkSparseImageOpaqueMemoryBindInfo*    pImageOpaqueBinds;
    uint32_t                                    imageBindCount;
    const VkSparseImageMemoryBindInfo*          pImageBinds;
    uint32_t                                    signalSemaphoreCount;
    const VkSemaphore*                          pSignalSemaphores;
} VkBindSparseInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `waitSemaphoreCount` is the number of semaphores upon which to wait before executing the sparse binding operations for the batch.
- `pWaitSemaphores` is a pointer to an array of semaphores upon which to wait on before the sparse binding operations for this batch begin execution. If semaphores to wait on are provided, they define a semaphore wait operation.
- `bufferBindCount` is the number of sparse buffer bindings to perform in the batch.
- `pBufferBinds` is a pointer to an array of VkSparseBufferMemoryBindInfo structures.
- `imageOpaqueBindCount` is the number of opaque sparse image bindings to perform.
- `pImageOpaqueBinds` is a pointer to an array of VkSparseImageOpaqueMemoryBindInfo structures, indicating opaque sparse image bindings to perform.
- `imageBindCount` is the number of sparse image bindings to perform.
- `pImageBinds` is a pointer to an array of VkSparseImageMemoryBindInfo structures, indicating sparse image bindings to perform.
- `signalSemaphoreCount` is the number of semaphores to be signaled once the sparse binding

operations specified by the structure have completed execution.

- pSignalSemaphores is a pointer to an array of semaphores which will be signaled when the sparse binding operations for this batch have completed execution. If semaphores to be signaled are provided, they define a semaphore signal operation.

Description

Valid Usage (Implicit)

- sType **must** be VK_STRUCTURE_TYPE_BIND_SPARSE_INFO
- pNext **must** be NULL
- If waitSemaphoreCount is not 0, pWaitSemaphores **must** be a pointer to an array of waitSemaphoreCount valid VkSemaphore handles
- If bufferBindCount is not 0, pBufferBinds **must** be a pointer to an array of bufferBindCount valid VkSparseBufferMemoryBindInfo structures
- If imageOpaqueBindCount is not 0, pImageOpaqueBinds **must** be a pointer to an array of imageOpaqueBindCount valid VkSparseImageOpaqueMemoryBindInfo structures
- If imageBindCount is not 0, pImageBinds **must** be a pointer to an array of imageBindCount valid VkSparseImageMemoryBindInfo structures
- If signalSemaphoreCount is not 0, pSignalSemaphores **must** be a pointer to an array of signalSemaphoreCount valid VkSemaphore handles
- Both of the elements of pSignalSemaphores, and the elements of pWaitSemaphores that are valid handles **must** have been created, allocated, or retrieved from the same VkDevice

See Also

VkSemaphore, VkSparseBufferMemoryBindInfo, VkSparseImageMemoryBindInfo, VkSparseImageOpaqueMemoryBindInfo, VkStructureType, vkQueueBindSparse

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBindSparseInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkBufferCopy(3)

Name

VkBufferCopy - Structure specifying a buffer copy operation

C Specification

The `VkBufferCopy` structure is defined as:

```
typedef struct VkBufferCopy {
    VkDeviceSize    srcOffset;
    VkDeviceSize    dstOffset;
    VkDeviceSize    size;
} VkBufferCopy;
```

Members

- `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.

Description

See Also

`VkDeviceSize`, vkCmdCopyBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBufferCopy

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkBufferCreateInfo(3)

Name

VkBufferCreateInfo - Structure specifying the parameters of a newly created buffer object

C Specification

The `VkBufferCreateInfo` structure is defined as:

```
typedef struct VkBufferCreateInfo {
    VkStructureType        sType;
    const void*            pNext;
    VkBufferCreateFlags    flags;
    VkDeviceSize           size;
    VkBufferUsageFlags     usage;
    VkSharingMode          sharingMode;
    uint32_t               queueFamilyIndexCount;
    const uint32_t*        pQueueFamilyIndices;
} VkBufferCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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 list of queue families that will access this buffer (ignored if `sharingMode` is not `VK_SHARING_MODE_CONCURRENT`).

Description

Valid Usage

- `size` **must** be greater than `0`
- If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, `pQueueFamilyIndices` **must** be a pointer to an array of `queueFamilyIndexCount` `uint32_t` values
- If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, `queueFamilyIndexCount` **must** be greater than `1`
- If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, each element of `pQueueFamilyIndices` **must** be unique and **must** be less than `pQueueFamilyPropertyCount` returned by vkGetPhysicalDeviceQueueFamilyProperties for the `physicalDevice` that was used to create `device`
- If the sparse bindings feature is not enabled, `flags` **must** not contain `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`
- If the sparse buffer residency feature is not enabled, `flags` **must** not contain `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT`
- If the sparse aliased residency feature is not enabled, `flags` **must** not contain `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT`
- If `flags` contains `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT` or `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT`, it **must** also contain `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be a valid combination of VkBufferCreateFlagBits values
- `usage` **must** be a valid combination of VkBufferUsageFlagBits values
- `usage` **must** not be `0`
- `sharingMode` **must** be a valid VkSharingMode value

See Also

VkBufferCreateFlags, VkBufferUsageFlags, `VkDeviceSize`, VkSharingMode, VkStructureType, vkCreateBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBufferCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkBufferImageCopy(3)

Name

VkBufferImageCopy - Structure specifying a buffer image copy operation

C Specification

For both vkCmdCopyBufferToImage and vkCmdCopyImageToBuffer, each element of `pRegions` is a structure defined as:

```
typedef struct VkBufferImageCopy {
    VkDeviceSize                bufferOffset;
    uint32_t                    bufferRowLength;
    uint32_t                    bufferImageHeight;
    VkImageSubresourceLayers    imageSubresource;
    VkOffset3D                  imageOffset;
    VkExtent3D                  imageExtent;
} VkBufferImageCopy;
```

Members

- `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 the data in buffer memory as a subregion of a larger two- or three-dimensional image, and control the addressing calculations of data in buffer memory. 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`.

Description

When copying to or from a depth or stencil aspect, the data in buffer memory uses a layout that is a (mostly) tightly packed representation of the depth or stencil data. Specifically:

- data copied to or from the stencil aspect of any depth/stencil format is tightly packed with one `VK_FORMAT_S8_UINT` value per texel.
- data copied to or from the depth aspect of a `VK_FORMAT_D16_UNORM` or `VK_FORMAT_D16_UNORM_S8_UINT` format is tightly packed with one `VK_FORMAT_D16_UNORM` value per texel.
- data copied to or from the depth aspect of a `VK_FORMAT_D32_SFLOAT` or `VK_FORMAT_D32_SFLOAT_S8_UINT` format is tightly packed with one `VK_FORMAT_D32_SFLOAT` value per

texel.

- data copied to or from the depth aspect of a `VK_FORMAT_X8_D24_UNORM_PACK32` or `VK_FORMAT_D24_UNORM_S8_UINT` format is packed with one 32-bit word per texel with the D24 value in the LSBs of the word, and undefined values in the eight MSBs.

> *Note*
>
> To copy both the depth and stencil aspects of a depth/stencil format, two entries in `pRegions` **can** be used, where one specifies the depth aspect in `imageSubresource`, and the other specifies the stencil aspect.

Because depth or stencil aspect buffer to image copies **may** require format conversions on some implementations, they are not supported on queues that do not support graphics. When copying to a depth aspect, the data in buffer memory **must** be in the the range [0,1] or undefined results occur.

Copies are done layer by layer starting with image layer `baseArrayLayer` member of `imageSubresource`. `layerCount` layers are copied from the source image or to the destination image.

Valid Usage

- If the the calling command's `VkImage` parameter's format is not a depth/stencil format, then `bufferOffset` **must** be a multiple of the format's element size
- `bufferOffset` **must** be a multiple of `4`
- `bufferRowLength` **must** be `0`, or greater than or equal to the `width` member of `imageExtent`
- `bufferImageHeight` **must** be `0`, or greater than or equal to the `height` member of `imageExtent`
- `imageOffset.x` and (`imageExtent.width` + `imageOffset.x`) **must** both be greater than or equal to `0` and less than or equal to the image subresource width
- `imageOffset.y` and (imageExtent.height + `imageOffset.y`) **must** both be greater than or equal to `0` and less than or equal to the image subresource height
- If the calling command's `srcImage` (vkCmdCopyImageToBuffer) or `dstImage` (vkCmdCopyBufferToImage) is of type `VK_IMAGE_TYPE_1D`, then `imageOffset.y` **must** be `0` and `imageExtent.height` **must** be `1`.
- `imageOffset.z` and (imageExtent.depth + `imageOffset.z`) **must** both be greater than or equal to `0` and less than or equal to the image subresource depth
- If the calling command's `srcImage` (vkCmdCopyImageToBuffer) or `dstImage` (vkCmdCopyBufferToImage) is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then `imageOffset.z` **must** be `0` and `imageExtent.depth` **must** be `1`.
- If the calling command's `VkImage` parameter is a compressed format image, `bufferRowLength` **must** be a multiple of the compressed texel block width
- If the calling command's `VkImage` parameter is a compressed format image, `bufferImageHeight` **must** be a multiple of the compressed texel block height
- If the calling command's `VkImage` parameter is a compressed format image, all members of `imageOffset` **must** be a multiple of the corresponding dimensions of the compressed texel block
- If the calling command's `VkImage` parameter is a compressed format image, `bufferOffset` **must** be a multiple of the compressed texel block size in bytes
- If the calling command's `VkImage` parameter is a compressed format image, `imageExtent.width` **must** be a multiple of the compressed texel block width or (`imageExtent.width` + `imageOffset.x`) **must** equal the image subresource width
- If the calling command's `VkImage` parameter is a compressed format image, `imageExtent.height` **must** be a multiple of the compressed texel block height or (`imageExtent.height` + `imageOffset.y`) **must** equal the image subresource height
- If the calling command's `VkImage` parameter is a compressed format image, `imageExtent.depth` **must** be a multiple of the compressed texel block depth or (`imageExtent.depth` + `imageOffset.z`) **must** equal the image subresource depth
- `bufferOffset`, `bufferRowLength`, `bufferImageHeight` and all members of `imageOffset` and `imageExtent` **must** respect the image transfer granularity requirements of the queue family that it will be submitted against, as described in Physical Device Enumeration
- The `aspectMask` member of `imageSubresource` **must** specify aspects present in the calling

command's `VkImage` parameter

- The `aspectMask` member of `imageSubresource` **must** only have a single bit set
- If the calling command's `VkImage` parameter is of VkImageType `VK_IMAGE_TYPE_3D`, the `baseArrayLayer` and `layerCount` members of `imageSubresource` **must** be `0` and `1`, respectively
- When copying to the depth aspect of an image subresource, the data in the source buffer **must** be in the range [0,1]

Valid Usage (Implicit)

- `imageSubresource` **must** be a valid `VkImageSubresourceLayers` structure

See Also

`VkDeviceSize`, VkExtent3D, VkImageSubresourceLayers, VkOffset3D, vkCmdCopyBufferToImage, vkCmdCopyImageToBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBufferImageCopy

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkBufferMemoryBarrier(3)

Name

VkBufferMemoryBarrier - Structure specifying a buffer memory barrier

C Specification

The `VkBufferMemoryBarrier` structure is defined as:

```
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;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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.

Description

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 `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 buffer 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 buffer range, and the second access scope includes no access, as if dstAccessMask was 0.

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 access scope includes no access, as if srcAccessMask was 0.

Valid Usage

- offset **must** be less than the size of buffer
- If size is not equal to VK_WHOLE_SIZE, size **must** be greater than 0
- If size is not equal to VK_WHOLE_SIZE, size **must** be less than or equal to than the size of buffer minus offset
- If buffer was created with a sharing mode of VK_SHARING_MODE_CONCURRENT, srcQueueFamilyIndex and dstQueueFamilyIndex **must** both be VK_QUEUE_FAMILY_IGNORED
- If buffer was created with a sharing mode of VK_SHARING_MODE_EXCLUSIVE, srcQueueFamilyIndex and dstQueueFamilyIndex **must** either both be VK_QUEUE_FAMILY_IGNORED, or both be a valid queue family (see html/vkspec.html#devsandqueues-queueprops)
- If buffer was created with a sharing mode of VK_SHARING_MODE_EXCLUSIVE, and srcQueueFamilyIndex and dstQueueFamilyIndex are not VK_QUEUE_FAMILY_IGNORED, at least one of them **must** be the same as the family of the queue that will execute this barrier

Valid Usage (Implicit)

- sType **must** be VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER
- pNext **must** be NULL
- srcAccessMask **must** be a valid combination of VkAccessFlagBits values
- dstAccessMask **must** be a valid combination of VkAccessFlagBits values
- buffer **must** be a valid VkBuffer handle

See Also

VkAccessFlags, VkBuffer, VkDeviceSize, VkStructureType, vkCmdPipelineBarrier, vkCmdWaitEvents

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBufferMemoryBarrier

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkBufferViewCreateInfo(3)

Name

VkBufferViewCreateInfo - Structure specifying parameters of a newly created buffer view

C Specification

The `VkBufferViewCreateInfo` structure is defined as:

```
typedef struct VkBufferViewCreateInfo {
    VkStructureType            sType;
    const void*                pNext;
    VkBufferViewCreateFlags    flags;
    VkBuffer                   buffer;
    VkFormat                   format;
    VkDeviceSize               offset;
    VkDeviceSize               range;
} VkBufferViewCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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 element size of `format`, then the nearest smaller multiple is used.

Description

Valid Usage

- `offset` **must** be less than the size of `buffer`
- `offset` **must** be a multiple of `VkPhysicalDeviceLimits`::`minTexelBufferOffsetAlignment`
- If `range` is not equal to `VK_WHOLE_SIZE`, `range` **must** be greater than `0`
- If `range` is not equal to `VK_WHOLE_SIZE`, `range` **must** be a multiple of the element size of `format`
- If `range` is not equal to `VK_WHOLE_SIZE`, `range` divided by the element size of `format` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxTexelBufferElements`
- 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`
- `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`
- If `buffer` was created with `usage` containing `VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT`, `format` **must** be supported for uniform texel buffers, as specified by the `VK_FORMAT_FEATURE_UNIFORM_TEXEL_BUFFER_BIT` flag in `VkFormatProperties`::`bufferFeatures` returned by `vkGetPhysicalDeviceFormatProperties`
- If `buffer` was created with `usage` containing `VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT`, `format` **must** be supported for storage texel buffers, as specified by the `VK_FORMAT_FEATURE_STORAGE_TEXEL_BUFFER_BIT` flag in `VkFormatProperties`::`bufferFeatures` returned by `vkGetPhysicalDeviceFormatProperties`
- If `buffer` is non-sparse then it **must** be bound completely and contiguously to a single `VkDeviceMemory` object

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_BUFFER_VIEW_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `buffer` **must** be a valid `VkBuffer` handle
- `format` **must** be a valid VkFormat value

See Also

VkBuffer, VkBufferViewCreateFlags, `VkDeviceSize`, VkFormat, VkStructureType, vkCreateBufferView

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBufferViewCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkClearAttachment(3)

Name

VkClearAttachment - Structure specifying a clear attachment

C Specification

The VkClearAttachment structure is defined as:

```
typedef struct VkClearAttachment {
    VkImageAspectFlags    aspectMask;
    uint32_t              colorAttachment;
    VkClearValue          clearValue;
} VkClearAttachment;
```

Members

- aspectMask is a mask selecting the color, depth and/or stencil aspects of the attachment to be cleared. aspectMask **can** include VK_IMAGE_ASPECT_COLOR_BIT for color attachments, VK_IMAGE_ASPECT_DEPTH_BIT for depth/stencil attachments with a depth component, and VK_IMAGE_ASPECT_STENCIL_BIT for depth/stencil attachments with a stencil component. If the subpass's depth/stencil attachment is VK_ATTACHMENT_UNUSED, then the clear has no effect.
- colorAttachment is only meaningful if VK_IMAGE_ASPECT_COLOR_BIT is set in aspectMask, in which case it is an index to the pColorAttachments array in the VkSubpassDescription structure of the current subpass which selects the color attachment to clear. If colorAttachment is VK_ATTACHMENT_UNUSED then the clear has no effect.
- clearValue is the color or depth/stencil value to clear the attachment to, as described in Clear Values below.

Description

No memory barriers are needed between vkCmdClearAttachments and preceding or subsequent draw or attachment clear commands in the same subpass.

The vkCmdClearAttachments command is not affected by the bound pipeline state.

Attachments **can** also be cleared at the beginning of a render pass instance by setting loadOp (or stencilLoadOp) of VkAttachmentDescription to VK_ATTACHMENT_LOAD_OP_CLEAR, as described for vkCreateRenderPass.

Valid Usage

- If `aspectMask` includes `VK_IMAGE_ASPECT_COLOR_BIT`, it **must** not include `VK_IMAGE_ASPECT_DEPTH_BIT` or `VK_IMAGE_ASPECT_STENCIL_BIT`
- `aspectMask` **must** not include `VK_IMAGE_ASPECT_METADATA_BIT`
- `clearValue` **must** be a valid `VkClearValue` union

Valid Usage (Implicit)

- `aspectMask` **must** be a valid combination of VkImageAspectFlagBits values
- `aspectMask` **must** not be `0`

See Also

VkClearValue, VkImageAspectFlags, vkCmdClearAttachments

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkClearAttachment

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkClearColorValue(3)

Name

VkClearColorValue - Structure specifying a clear color value

C Specification

The `VkClearColorValue` structure is defined as:

```
typedef union VkClearColorValue {
    float       float32[4];
    int32_t     int32[4];
    uint32_t    uint32[4];
} VkClearColorValue;
```

Members

- `float32` are the color clear values when the format of the image or attachment is one of the formats in the Interpretation of Numeric Format table other than signed integer (`SINT`) or unsigned integer (`UINT`). 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 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 is unsigned integer (`UINT`). Unsigned integer values are converted to the format of the image by casting to the integer type with fewer bits.

Description

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.

See Also

VkClearValue, vkCmdClearColorImage

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkClearColorValue

VkClearDepthStencilValue(3)

Name

VkClearDepthStencilValue - Structure specifying a clear depth stencil value

C Specification

The `VkClearDepthStencilValue` structure is defined as:

```
typedef struct VkClearDepthStencilValue {
    float       depth;
    uint32_t    stencil;
} VkClearDepthStencilValue;
```

Members

- `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.

Description

Valid Usage

- `depth` **must** be between `0.0` and `1.0`, inclusive

See Also

VkClearValue, vkCmdClearDepthStencilImage

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkClearDepthStencilValue

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkClearRect(3)

Name

VkClearRect - Structure specifying a clear rectangle

C Specification

The `VkClearRect` structure is defined as:

```
typedef struct VkClearRect {
    VkRect2D    rect;
    uint32_t    baseArrayLayer;
    uint32_t    layerCount;
} VkClearRect;
```

Members

- `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.

Description

The layers [`baseArrayLayer`, `baseArrayLayer` + `layerCount`) counting from the base layer of the attachment image view are cleared.

See Also

VkRect2D, vkCmdClearAttachments

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkClearRect

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkClearValue(3)

Name

VkClearValue - Structure specifying a clear value

C Specification

The `VkClearValue` union is defined as:

```
typedef union VkClearValue {
    VkClearColorValue           color;
    VkClearDepthStencilValue    depthStencil;
} VkClearValue;
```

Members

- `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.

Description

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.

Valid Usage

- `depthStencil` **must** be a valid `VkClearDepthStencilValue` structure

See Also

VkClearAttachment, VkClearColorValue, VkClearDepthStencilValue, VkRenderPassBeginInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkClearValue

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCommandBufferAllocateInfo(3)

Name

VkCommandBufferAllocateInfo - Structure specifying the allocation parameters for command buffer object

C Specification

The `VkCommandBufferAllocateInfo` structure is defined as:

```
typedef struct VkCommandBufferAllocateInfo {
    VkStructureType         sType;
    const void*             pNext;
    VkCommandPool           commandPool;
    VkCommandBufferLevel    level;
    uint32_t                commandBufferCount;
} VkCommandBufferAllocateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `commandPool` is the command pool from which the command buffers are allocated.
- `level` is an VkCommandBufferLevel value specifying the command buffer level.
- `commandBufferCount` is the number of command buffers to allocate from the pool.

Description

Valid Usage

- `commandBufferCount` **must** be greater than `0`

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO`
- `pNext` **must** be `NULL`
- `commandPool` **must** be a valid `VkCommandPool` handle
- `level` **must** be a valid VkCommandBufferLevel value

See Also

VkCommandBufferLevel, VkCommandPool, VkStructureType, vkAllocateCommandBuffers

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandBufferAllocateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCommandBufferBeginInfo(3)

Name

VkCommandBufferBeginInfo - Structure specifying a command buffer begin operation

C Specification

The VkCommandBufferBeginInfo structure is defined as:

```
typedef struct VkCommandBufferBeginInfo {
    VkStructureType                          sType;
    const void*                              pNext;
    VkCommandBufferUsageFlags                flags;
    const VkCommandBufferInheritanceInfo*    pInheritanceInfo;
} VkCommandBufferBeginInfo;
```

Members

- sType is the type of this structure.
- pNext is NULL or a pointer to an extension-specific structure.
- flags is a bitmask of VkCommandBufferUsageFlagBits specifying usage behavior for the command buffer.
- pInheritanceInfo is a pointer to a VkCommandBufferInheritanceInfo structure, which is used if commandBuffer is a secondary command buffer. If this is a primary command buffer, then this value is ignored.

Description

Valid Usage

- If flags contains VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT, the renderPass member of pInheritanceInfo **must** be a valid VkRenderPass
- 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
- If flags contains VK_COMMAND_BUFFER_USAGE_RENDER_PASS_CONTINUE_BIT, the framebuffer member of pInheritanceInfo **must** be either VK_NULL_HANDLE, or a valid VkFramebuffer that is compatible with the renderPass member of pInheritanceInfo

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be a valid combination of VkCommandBufferUsageFlagBits values

See Also

VkCommandBufferInheritanceInfo, VkCommandBufferUsageFlags, VkStructureType, vkBeginCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandBufferBeginInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCommandBufferInheritanceInfo(3)

Name

VkCommandBufferInheritanceInfo - Structure specifying command buffer inheritance info

C Specification

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:

```
typedef struct VkCommandBufferInheritanceInfo {
    VkStructureType                  sType;
    const void*                      pNext;
    VkRenderPass                     renderPass;
    uint32_t                         subpass;
    VkFramebuffer                    framebuffer;
    VkBool32                         occlusionQueryEnable;
    VkQueryControlFlags              queryFlags;
    VkQueryPipelineStatisticFlags    pipelineStatistics;
} VkCommandBufferInheritanceInfo;
```

Members

- sType is the type of this structure.
- pNext is NULL or a pointer to an extension-specific structure.
- renderPass is a VkRenderPass object defining which render passes the VkCommandBuffer will be compatible with and **can** be executed within. If the VkCommandBuffer will not be executed within a render pass instance, renderPass is ignored.
- subpass is the index of the subpass within the render pass instance that the VkCommandBuffer will be executed within. If the VkCommandBuffer will not be executed within a render pass instance, subpass is ignored.
- framebuffer optionally refers 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, or if the VkCommandBuffer will not be executed within a render pass instance.

> *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 indicates 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 indicates 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.

Description

Valid Usage

- If the inherited queries feature is not enabled, occlusionQueryEnable **must** be VK_FALSE
- If the inherited queries feature is enabled, queryFlags **must** be a valid combination of VkQueryControlFlagBits values
- If the pipeline statistics queries feature is not enabled, pipelineStatistics **must** be 0

Valid Usage (Implicit)

- sType **must** be VK_STRUCTURE_TYPE_COMMAND_BUFFER_INHERITANCE_INFO
- pNext **must** be NULL
- Both of framebuffer, and renderPass that are valid handles **must** have been created, allocated, or retrieved from the same VkDevice

See Also

VkBool32, VkCommandBufferBeginInfo, VkFramebuffer, VkQueryControlFlags, VkQueryPipelineStatisticFlags, VkRenderPass, VkStructureType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandBufferInheritanceInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCommandPoolCreateInfo(3)

Name

VkCommandPoolCreateInfo - Structure specifying parameters of a newly created command pool

C Specification

The `VkCommandPoolCreateInfo` structure is defined as:

```
typedef struct VkCommandPoolCreateInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkCommandPoolCreateFlags    flags;
    uint32_t                    queueFamilyIndex;
} VkCommandPoolCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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.

Description

Valid Usage

- `queueFamilyIndex` **must** be the index of a queue family available in the calling command's `device` parameter

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be a valid combination of VkCommandPoolCreateFlagBits values

See Also

VkCommandPoolCreateFlags, VkStructureType, vkCreateCommandPool

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandPoolCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkComponentMapping(3)

Name

VkComponentMapping - Structure specifying a color component mapping

C Specification

The `VkComponentMapping` structure is defined as:

```
typedef struct VkComponentMapping {
    VkComponentSwizzle    r;
    VkComponentSwizzle    g;
    VkComponentSwizzle    b;
    VkComponentSwizzle    a;
} VkComponentMapping;
```

Members

- `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.

Description

Valid Usage (Implicit)

- `r` **must** be a valid VkComponentSwizzle value
- `g` **must** be a valid VkComponentSwizzle value
- `b` **must** be a valid VkComponentSwizzle value
- `a` **must** be a valid VkComponentSwizzle value

See Also

VkComponentSwizzle, VkImageViewCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkComponentMapping

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkComputePipelineCreateInfo(3)

Name

VkComputePipelineCreateInfo - Structure specifying parameters of a newly created compute pipeline

C Specification

The `VkComputePipelineCreateInfo` structure is defined as:

```
typedef struct VkComputePipelineCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    VkPipelineCreateFlags              flags;
    VkPipelineShaderStageCreateInfo    stage;
    VkPipelineLayout                   layout;
    VkPipeline                         basePipelineHandle;
    int32_t                            basePipelineIndex;
} VkComputePipelineCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is a bitmask of VkPipelineCreateFlagBits specifying how the pipeline will be generated.
- `stage` is a VkPipelineShaderStageCreateInfo 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
- `basePipelineIndex` is an index into the `pCreateInfos` parameter to use as a pipeline to derive from

Description

The parameters `basePipelineHandle` and `basePipelineIndex` are described in more detail in Pipeline Derivatives.

`stage` points to a structure of type `VkPipelineShaderStageCreateInfo`.

Valid Usage

- If `flags` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, and `basePipelineIndex` is -1, `basePipelineHandle` **must** be a valid handle to a compute `VkPipeline`
- If `flags` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, and `basePipelineHandle` is VK_NULL_HANDLE, `basePipelineIndex` **must** be a valid index into the calling command's `pCreateInfos` parameter
- If `flags` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, and `basePipelineIndex` is not -1, `basePipelineHandle` **must** be VK_NULL_HANDLE
- If `flags` contains the `VK_PIPELINE_CREATE_DERIVATIVE_BIT` flag, and `basePipelineHandle` is not VK_NULL_HANDLE, `basePipelineIndex` **must** be -1
- The `stage` member of `stage` **must** be `VK_SHADER_STAGE_COMPUTE_BIT`
- 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
- `layout` **must** be consistent with the layout of the compute shader specified in `stage`

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be a valid combination of VkPipelineCreateFlagBits values
- `stage` **must** be a valid `VkPipelineShaderStageCreateInfo` structure
- `layout` **must** be a valid `VkPipelineLayout` handle
- Both of `basePipelineHandle`, and `layout` that are valid handles **must** have been created, allocated, or retrieved from the same `VkDevice`

See Also

VkPipeline, VkPipelineCreateFlags, VkPipelineLayout, VkPipelineShaderStageCreateInfo, VkStructureType, vkCreateComputePipelines

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkComputePipelineCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCopyDescriptorSet(3)

Name

VkCopyDescriptorSet - Structure specifying a copy descriptor set operation

C Specification

The `VkCopyDescriptorSet` structure is defined as:

```
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;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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.

Description

Valid Usage

- `srcBinding` **must** be a valid binding within `srcSet`
- 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, as described by html/vkspec.html#descriptorsets-updates-consecutive
- `dstBinding` **must** be a valid binding within `dstSet`
- 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, as described by html/vkspec.html#descriptorsets-updates-consecutive
- 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 as described by html/vkspec.html#descriptorsets-updates-consecutive

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_COPY_DESCRIPTOR_SET`
- `pNext` **must** be `NULL`
- `srcSet` **must** be a valid `VkDescriptorSet` handle
- `dstSet` **must** be a valid `VkDescriptorSet` handle
- Both of `dstSet`, and `srcSet` **must** have been created, allocated, or retrieved from the same `VkDevice`

See Also

VkDescriptorSet, VkStructureType, vkUpdateDescriptorSets

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCopyDescriptorSet

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorBufferInfo(3)

Name

VkDescriptorBufferInfo - Structure specifying descriptor buffer info

C Specification

The `VkDescriptorBufferInfo` structure is defined as:

```
typedef struct VkDescriptorBufferInfo {
    VkBuffer        buffer;
    VkDeviceSize    offset;
    VkDeviceSize    range;
} VkDescriptorBufferInfo;
```

Members

- `buffer` is 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.

Description

> *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.

Valid Usage

- `offset` **must** be less than the size of `buffer`
- If `range` is not equal to `VK_WHOLE_SIZE`, `range` **must** be greater than `0`
- If `range` is not equal to `VK_WHOLE_SIZE`, `range` **must** be less than or equal to the size of `buffer` minus `offset`

Valid Usage (Implicit)

- `buffer` **must** be a valid `VkBuffer` handle

See Also

VkBuffer, `VkDeviceSize`, VkWriteDescriptorSet

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorBufferInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorImageInfo(3)

Name

VkDescriptorImageInfo - Structure specifying descriptor image info

C Specification

The `VkDescriptorImageInfo` structure is defined as:

```
typedef struct VkDescriptorImageInfo {
    VkSampler        sampler;
    VkImageView      imageView;
    VkImageLayout    imageLayout;
} VkDescriptorImageInfo;
```

Members

- `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 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`.

Description

Members of `VkDescriptorImageInfo` that are not used in an update (as described above) are ignored.

Valid Usage

- `imageLayout` **must** match the actual VkImageLayout of each subresource accessible from `imageView` at the time this descriptor is accessed

Valid Usage (Implicit)

- Both of `imageView`, and `sampler` that are valid handles **must** have been created, allocated, or retrieved from the same `VkDevice`

See Also

VkImageLayout, VkImageView, VkSampler, VkWriteDescriptorSet

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorImageInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorPoolCreateInfo(3)

Name

VkDescriptorPoolCreateInfo - Structure specifying parameters of a newly created descriptor pool

C Specification

Additional information about the pool is passed in an instance of the `VkDescriptorPoolCreateInfo` structure:

```
typedef struct VkDescriptorPoolCreateInfo {
    VkStructureType                sType;
    const void*                    pNext;
    VkDescriptorPoolCreateFlags    flags;
    uint32_t                       maxSets;
    uint32_t                       poolSizeCount;
    const VkDescriptorPoolSize*    pPoolSizes;
} VkDescriptorPoolCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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.

Description

If multiple `VkDescriptorPoolSize` structures 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.

Valid Usage

- `maxSets` **must** be greater than `0`

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be a valid combination of VkDescriptorPoolCreateFlagBits values
- `pPoolSizes` **must** be a pointer to an array of `poolSizeCount` valid `VkDescriptorPoolSize` structures
- `poolSizeCount` **must** be greater than `0`

See Also

VkDescriptorPoolCreateFlags, VkDescriptorPoolSize, VkStructureType, vkCreateDescriptorPool

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorPoolCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorPoolSize(3)

Name

VkDescriptorPoolSize - Structure specifying descriptor pool size

C Specification

The `VkDescriptorPoolSize` structure is defined as:

```
typedef struct VkDescriptorPoolSize {
    VkDescriptorType    type;
    uint32_t            descriptorCount;
} VkDescriptorPoolSize;
```

Members

- `type` is the type of descriptor.
- `descriptorCount` is the number of descriptors of that type to allocate.

Description

Valid Usage

- `descriptorCount` **must** be greater than `0`

Valid Usage (Implicit)

- `type` **must** be a valid VkDescriptorType value

See Also

VkDescriptorPoolCreateInfo, VkDescriptorType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorPoolSize

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorSetAllocateInfo(3)

Name

VkDescriptorSetAllocateInfo - Structure specifying the allocation parameters for descriptor sets

C Specification

The `VkDescriptorSetAllocateInfo` structure is defined as:

```
typedef struct VkDescriptorSetAllocateInfo {
    VkStructureType                 sType;
    const void*                     pNext;
    VkDescriptorPool                descriptorPool;
    uint32_t                        descriptorSetCount;
    const VkDescriptorSetLayout*    pSetLayouts;
} VkDescriptorSetAllocateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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 an array of descriptor set layouts, with each member specifying how the corresponding descriptor set is allocated.

Description

Valid Usage

- `descriptorSetCount` **must** not be greater than the number of sets that are currently available for allocation in `descriptorPool`
- `descriptorPool` **must** have enough free descriptor capacity remaining to allocate the descriptor sets of the specified layouts

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO`
- `pNext` **must** be `NULL`
- `descriptorPool` **must** be a valid `VkDescriptorPool` handle
- `pSetLayouts` **must** be a pointer to an array of `descriptorSetCount` valid `VkDescriptorSetLayout` handles
- `descriptorSetCount` **must** be greater than `0`
- Both of `descriptorPool`, and the elements of `pSetLayouts` **must** have been created, allocated, or retrieved from the same `VkDevice`

See Also

VkDescriptorPool, VkDescriptorSetLayout, VkStructureType, vkAllocateDescriptorSets

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorSetAllocateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorSetLayoutBinding(3)

Name

VkDescriptorSetLayoutBinding - Structure specifying a descriptor set layout binding

C Specification

The VkDescriptorSetLayoutBinding structure is defined as:

```
typedef struct VkDescriptorSetLayoutBinding {
    uint32_t              binding;
    VkDescriptorType      descriptorType;
    uint32_t              descriptorCount;
    VkShaderStageFlags    stageFlags;
    const VkSampler*      pImmutableSamplers;
} VkDescriptorSetLayoutBinding;
```

Members

- 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. There are no limitations on what combinations of stages **can** be used by a descriptor binding, and in particular a binding **can** be used by both graphics stages and the compute stage.

Description

- 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; later binding a sampler into an immutable sampler slot in a descriptor set is not allowed. If pImmutableSamplers is not NULL, then it is considered to be a pointer to an array of sampler handles that will be consumed by the set layout and used for the corresponding binding. 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 `descriptorType` is treated as 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

- 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 pointer to an array of `descriptorCount` valid `VkSampler` handles
- If `descriptorCount` is not `0`, `stageFlags` **must** be a valid combination of VkShaderStageFlagBits values

Valid Usage (Implicit)

- `descriptorType` **must** be a valid VkDescriptorType value

See Also

VkDescriptorSetLayoutCreateInfo, VkDescriptorType, VkSampler, VkShaderStageFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorSetLayoutBinding

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorSetLayoutCreateInfo(3)

Name

VkDescriptorSetLayoutCreateInfo - Structure specifying parameters of a newly created descriptor set layout

C Specification

Information about the descriptor set layout is passed in an instance of the `VkDescriptorSetLayoutCreateInfo` structure:

```
typedef struct VkDescriptorSetLayoutCreateInfo {
    VkStructureType                        sType;
    const void*                            pNext;
    VkDescriptorSetLayoutCreateFlags       flags;
    uint32_t                               bindingCount;
    const VkDescriptorSetLayoutBinding*    pBindings;
} VkDescriptorSetLayoutCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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.

Description

Valid Usage

- The VkDescriptorSetLayoutBinding::`binding` members of the elements of the `pBindings` array **must** each have different values.

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be a valid combination of VkDescriptorSetLayoutCreateFlagBits values
- If `bindingCount` is not `0`, `pBindings` **must** be a pointer to an array of `bindingCount` valid `VkDescriptorSetLayoutBinding` structures

See Also

VkDescriptorSetLayoutBinding, VkDescriptorSetLayoutCreateFlags, VkStructureType, vkCreateDescriptorSetLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorSetLayoutCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDeviceCreateInfo(3)

Name

VkDeviceCreateInfo - Structure specifying parameters of a newly created device

C Specification

The `VkDeviceCreateInfo` structure is defined as:

```
typedef struct VkDeviceCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    VkDeviceCreateFlags                flags;
    uint32_t                           queueCreateInfoCount;
    const VkDeviceQueueCreateInfo*     pQueueCreateInfos;
    uint32_t                           enabledLayerCount;
    const char* const*                 ppEnabledLayerNames;
    uint32_t                           enabledExtensionCount;
    const char* const*                 ppEnabledExtensionNames;
    const VkPhysicalDeviceFeatures*    pEnabledFeatures;
} VkDeviceCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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 deprecated and ignored.
- `ppEnabledLayerNames` is deprecated and ignored. See Device Layer Deprecation.
- `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 that contains boolean indicators of all the features to be enabled. Refer to the Features section for further details.

Description

Valid Usage

- The `queueFamilyIndex` member of any given element of `pQueueCreateInfos` **must** be unique within `pQueueCreateInfos`

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `pQueueCreateInfos` **must** be a pointer to an array of `queueCreateInfoCount` valid `VkDeviceQueueCreateInfo` structures
- If `enabledLayerCount` is not `0`, `ppEnabledLayerNames` **must** be a pointer to an array of `enabledLayerCount` null-terminated UTF-8 strings
- If `enabledExtensionCount` is not `0`, `ppEnabledExtensionNames` **must** be a pointer to an array of `enabledExtensionCount` null-terminated UTF-8 strings
- If `pEnabledFeatures` is not `NULL`, `pEnabledFeatures` **must** be a pointer to a valid `VkPhysicalDeviceFeatures` structure
- `queueCreateInfoCount` **must** be greater than `0`

See Also

VkDeviceCreateFlags, VkDeviceQueueCreateInfo, VkPhysicalDeviceFeatures, VkStructureType, vkCreateDevice

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDeviceCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDeviceQueueCreateInfo(3)

Name

VkDeviceQueueCreateInfo - Structure specifying parameters of a newly created device queue

C Specification

The `VkDeviceQueueCreateInfo` structure is defined as:

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

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `queueFamilyIndex` is an unsigned integer indicating the index of the queue family to create 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`.
- `pQueuePriorities` is 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.

Description

Valid Usage

- `queueFamilyIndex` **must** be less than `pQueueFamilyPropertyCount` returned by `vkGetPhysicalDeviceQueueFamilyProperties`
- `queueCount` **must** be less than or equal to the `queueCount` member of the `VkQueueFamilyProperties` structure, as returned by `vkGetPhysicalDeviceQueueFamilyProperties` in the `pQueueFamilyProperties[queueFamilyIndex]`
- Each element of `pQueuePriorities` **must** be between `0.0` and `1.0` inclusive

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `pQueuePriorities` **must** be a pointer to an array of `queueCount` `float` values
- `queueCount` **must** be greater than `0`

See Also

VkDeviceCreateInfo, VkDeviceQueueCreateFlags, VkStructureType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDeviceQueueCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDispatchIndirectCommand(3)

Name

VkDispatchIndirectCommand - Structure specifying a dispatch indirect command

C Specification

The `VkDispatchIndirectCommand` structure is defined as:

```
typedef struct VkDispatchIndirectCommand {
    uint32_t    x;
    uint32_t    y;
    uint32_t    z;
} VkDispatchIndirectCommand;
```

Members

- `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.

Description

The members of `VkDispatchIndirectCommand` have the same meaning as the corresponding parameters of vkCmdDispatch.

Valid Usage

- `x` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[0]
- `y` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[1]
- `z` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxComputeWorkGroupCount`[2]

See Also

vkCmdDispatchIndirect

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDispatchIndirectCommand

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDrawIndexedIndirectCommand(3)

Name

VkDrawIndexedIndirectCommand - Structure specifying a draw indexed indirect command

C Specification

The `VkDrawIndexedIndirectCommand` structure is defined as:

```
typedef struct VkDrawIndexedIndirectCommand {
    uint32_t    indexCount;
    uint32_t    instanceCount;
    uint32_t    firstIndex;
    int32_t     vertexOffset;
    uint32_t    firstInstance;
} VkDrawIndexedIndirectCommand;
```

Members

- `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.

Description

The members of `VkDrawIndexedIndirectCommand` have the same meaning as the similarly named parameters of vkCmdDrawIndexed.

Valid Usage

- For a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in html/vkspec.html#fxvertex-input
- (`indexSize` * (`firstIndex` + `indexCount`) + `offset`) **must** be less than or equal to the size of the currently bound index buffer, with `indexSize` being based on the type specified by `indexType`, where the index buffer, `indexType`, and `offset` are specified via `vkCmdBindIndexBuffer`
- If the drawIndirectFirstInstance feature is not enabled, `firstInstance` **must** be `0`

See Also

vkCmdDrawIndexedIndirect

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDrawIndexedIndirectCommand

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDrawIndirectCommand(3)

Name

VkDrawIndirectCommand - Structure specifying a draw indirect command

C Specification

The `VkDrawIndirectCommand` structure is defined as:

```
typedef struct VkDrawIndirectCommand {
    uint32_t    vertexCount;
    uint32_t    instanceCount;
    uint32_t    firstVertex;
    uint32_t    firstInstance;
} VkDrawIndirectCommand;
```

Members

- `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.

Description

The members of `VkDrawIndirectCommand` have the same meaning as the similarly named parameters of vkCmdDraw.

Valid Usage

- For a given vertex buffer binding, any attribute data fetched **must** be entirely contained within the corresponding vertex buffer binding, as described in html/vkspec.html#fxvertex-input
- If the drawIndirectFirstInstance feature is not enabled, `firstInstance` **must** be `0`

See Also

vkCmdDrawIndirect

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDrawIndirectCommand

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkEventCreateInfo(3)

Name

VkEventCreateInfo - Structure specifying parameters of a newly created event

C Specification

The `VkEventCreateInfo` structure is defined as:

```
typedef struct VkEventCreateInfo {
    VkStructureType       sType;
    const void*           pNext;
    VkEventCreateFlags    flags;
} VkEventCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.

Description

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_EVENT_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`

See Also

VkEventCreateFlags, VkStructureType, vkCreateEvent

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkEventCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkExtensionProperties(3)

Name

VkExtensionProperties - Structure specifying a extension properties

C Specification

The `VkExtensionProperties` structure is defined as:

```
typedef struct VkExtensionProperties {
    char        extensionName[VK_MAX_EXTENSION_NAME_SIZE];
    uint32_t    specVersion;
} VkExtensionProperties;
```

Members

- `extensionName` is a null-terminated string specifying the name of the extension.
- `specVersion` is the version of this extension. It is an integer, incremented with backward compatible changes.

Description

See Also

vkEnumerateDeviceExtensionProperties, vkEnumerateInstanceExtensionProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkExtensionProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkExtent2D(3)

Name

VkExtent2D - Structure specifying a two-dimensional extent

C Specification

A two-dimensional extent is defined by the structure:

```
typedef struct VkExtent2D {
    uint32_t    width;
    uint32_t    height;
} VkExtent2D;
```

Members

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

Description

See Also

VkRect2D, vkGetRenderAreaGranularity

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkExtent2D

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkExtent3D(3)

Name

VkExtent3D - Structure specifying a three-dimensional extent

C Specification

A three-dimensional extent is defined by the structure:

```
typedef struct VkExtent3D {
    uint32_t    width;
    uint32_t    height;
    uint32_t    depth;
} VkExtent3D;
```

Members

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

Description

See Also

VkBufferImageCopy, VkImageCopy, VkImageCreateInfo, VkImageFormatProperties, VkImageResolve, VkQueueFamilyProperties, VkSparseImageFormatProperties, VkSparseImageMemoryBind

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkExtent3D

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkFenceCreateInfo(3)

Name

VkFenceCreateInfo - Structure specifying parameters of a newly created fence

C Specification

The `VkFenceCreateInfo` structure is defined as:

```
typedef struct VkFenceCreateInfo {
    VkStructureType       sType;
    const void*           pNext;
    VkFenceCreateFlags    flags;
} VkFenceCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is a bitmask of VkFenceCreateFlagBits specifying the initial state and behavior of the fence.

Description

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_FENCE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be a valid combination of VkFenceCreateFlagBits values

See Also

VkFenceCreateFlags, VkStructureType, vkCreateFence

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFenceCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkFormatProperties(3)

Name

VkFormatProperties - Structure specifying image format properties

C Specification

The `VkFormatProperties` structure is defined as:

```
typedef struct VkFormatProperties {
    VkFormatFeatureFlags    linearTilingFeatures;
    VkFormatFeatureFlags    optimalTilingFeatures;
    VkFormatFeatureFlags    bufferFeatures;
} VkFormatProperties;
```

Members

- `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.

Description

Note

If no format feature flags are supported, then the only possible use would be image transfers - which alone are not useful. As such, if no format feature flags are supported, the format itself is not supported, and images of that format cannot be created.

If `format` is a block-compression format, then buffers **must** not support any features for the format.

See Also

VkFormatFeatureFlags, vkGetPhysicalDeviceFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFormatProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the

Specification, not directly.

VkFramebufferCreateInfo(3)

Name

VkFramebufferCreateInfo - Structure specifying parameters of a newly created framebuffer

C Specification

The `VkFramebufferCreateInfo` structure is defined as:

```
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;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `renderPass` is a render pass that defines what render passes the framebuffer will be compatible with. See Render Pass Compatibility for details.
- `attachmentCount` is the number of attachments.
- `pAttachments` is an array of `VkImageView` handles, each of which will be used as the corresponding attachment in a render pass instance.
- `width`, `height` and `layers` define the dimensions of the framebuffer.

Description

Image subresources used as attachments **must** not be used via any non-attachment usage for the duration of a render pass instance.

> *Note*
>
> This restriction means that the render pass has full knowledge of all uses of all of the attachments, so that the implementation is able to make correct decisions about when and how to perform layout transitions, when to overlap execution of subpasses, etc.

It is legal for a subpass to use no color or depth/stencil attachments, and rather 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 a given zero-attachment subpass **must** have the same value for VkPipelineMultisampleStateCreateInfo ::`rasterizationSamples`.

Valid Usage

- `attachmentCount` **must** be equal to the attachment count specified in `renderPass`
- Any given 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`
- Any given 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`
- Any given 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`
- Any given element of `pAttachments` **must** have been created with an VkFormat value that matches the VkFormat specified by the corresponding `VkAttachmentDescription` in `renderPass`
- Any given element of `pAttachments` **must** have been created with a `samples` value that matches the `samples` value specified by the corresponding `VkAttachmentDescription` in `renderPass`
- Any given element of `pAttachments` **must** have dimensions at least as large as the corresponding framebuffer dimension
- Any given element of `pAttachments` **must** only specify a single mip level
- Any given element of `pAttachments` **must** have been created with the identity swizzle
- `width` **must** be greater than `0`.
- `width` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxFramebufferWidth`
- `height` **must** be greater than `0`.
- `height` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxFramebufferHeight`
- `layers` **must** be greater than `0`.
- `layers` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxFramebufferLayers`

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `renderPass` **must** be a valid `VkRenderPass` handle
- If `attachmentCount` is not `0`, `pAttachments` **must** be a pointer to an array of `attachmentCount` valid `VkImageView` handles
- Both of `renderPass`, and the elements of `pAttachments` that are valid handles **must** have been created, allocated, or retrieved from the same `VkDevice`

See Also

VkFramebufferCreateFlags, VkImageView, VkRenderPass, VkStructureType, vkCreateFramebuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFramebufferCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkGraphicsPipelineCreateInfo(3)

Name

VkGraphicsPipelineCreateInfo - Structure specifying parameters of a newly created graphics pipeline

C Specification

The `VkGraphicsPipelineCreateInfo` structure is defined as:

```
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;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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 an array of size `stageCount` structures of type VkPipelineShaderStageCreateInfo describing the set of the shader stages to be included in the graphics pipeline.
- `pVertexInputState` is a pointer to an instance of the VkPipelineVertexInputStateCreateInfo structure.
- `pInputAssemblyState` is a pointer to an instance of the VkPipelineInputAssemblyStateCreateInfo structure which determines input assembly behavior, as described in Drawing Commands.

- pTessellationState is a pointer to an instance of the VkPipelineTessellationStateCreateInfo structure, and is ignored if the pipeline does not include a tessellation control shader stage and tessellation evaluation shader stage.
- pViewportState is a pointer to an instance of the VkPipelineViewportStateCreateInfo structure, and is ignored if the pipeline has rasterization disabled.
- pRasterizationState is a pointer to an instance of the VkPipelineRasterizationStateCreateInfo structure.
- pMultisampleState is a pointer to an instance of the VkPipelineMultisampleStateCreateInfo, and is ignored if the pipeline has rasterization disabled.
- pDepthStencilState is a pointer to an instance of the VkPipelineDepthStencilStateCreateInfo structure, and is ignored if the pipeline has rasterization disabled or if the subpass of the render pass the pipeline is created against does not use a depth/stencil attachment.
- pColorBlendState is a pointer to an instance of the VkPipelineColorBlendStateCreateInfo structure, and is ignored if the pipeline has rasterization disabled or if the subpass of the render pass the pipeline is created against does not use any color attachments.
- pDynamicState is a pointer to VkPipelineDynamicStateCreateInfo and is used to indicate 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 an instance of any render pass compatible with the one provided. See Render Pass Compatibility for more information.
- subpass is the index of the subpass in the render pass where this pipeline will be used.
- basePipelineHandle is a pipeline to derive from.
- basePipelineIndex is an index into the pCreateInfos parameter to use as a pipeline to derive from.

Description

The parameters basePipelineHandle and basePipelineIndex are described in more detail in Pipeline Derivatives.

pStages points to an array of VkPipelineShaderStageCreateInfo structures, which were previously described in Compute Pipelines.

pDynamicState points to a structure of type VkPipelineDynamicStateCreateInfo.

Valid Usage

- If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and basePipelineIndex is -1, basePipelineHandle **must** be a valid handle to a graphics VkPipeline
- If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and basePipelineHandle is VK_NULL_HANDLE, basePipelineIndex **must** be a valid index into the calling command's pCreateInfos parameter
- If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and basePipelineIndex is not -1, basePipelineHandle **must** be VK_NULL_HANDLE
- If flags contains the VK_PIPELINE_CREATE_DERIVATIVE_BIT flag, and basePipelineHandle is not VK_NULL_HANDLE, basePipelineIndex **must** be -1
- The stage member of each element of pStages **must** be unique
- The stage member of one element of pStages **must** be VK_SHADER_STAGE_VERTEX_BIT
- The stage member of any given element of pStages **must** not be VK_SHADER_STAGE_COMPUTE_BIT
- If pStages includes a tessellation control shader stage, it **must** include a tessellation evaluation shader stage
- If pStages includes a tessellation evaluation shader stage, it **must** include a tessellation control shader stage
- If pStages includes a tessellation control shader stage and a tessellation evaluation shader stage, pTessellationState **must** be a pointer to a valid VkPipelineTessellationStateCreateInfo structure
- If pStages includes tessellation shader stages, the shader code of at least one stage **must** contain an OpExecutionMode instruction that specifies the type of subdivision in the pipeline
- If pStages includes tessellation shader stages, and the shader code of both stages contain an OpExecutionMode instruction that specifies the type of subdivision in the pipeline, they **must** both specify the same subdivision mode
- If pStages includes tessellation shader stages, the shader code of at least one stage **must** contain an OpExecutionMode instruction that specifies the output patch size in the pipeline
- If pStages includes tessellation shader stages, and the shader code of both contain an OpExecutionMode instruction that specifies the out patch size in the pipeline, they **must** both specify the same patch size
- If pStages includes tessellation shader stages, the topology member of pInputAssembly **must** be VK_PRIMITIVE_TOPOLOGY_PATCH_LIST
- If the topology member of pInputAssembly is VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, pStages **must** include tessellation shader stages
- If pStages includes a geometry shader stage, and does not include any tessellation shader stages, its shader code **must** contain an OpExecutionMode instruction that specifies an input primitive type that is compatible with the primitive topology specified in pInputAssembly
- If pStages includes a geometry shader stage, and also includes tessellation shader stages,

its shader code **must** contain an OpExecutionMode instruction that specifies an input primitive type that is compatible with the primitive topology that is output by the tessellation stages

- If pStages includes a fragment shader stage and a geometry shader stage, 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
- If pStages includes a fragment shader stage, its shader code **must** not read from any input attachment that is defined as VK_ATTACHMENT_UNUSED in subpass
- The shader code for the entry points identified by pStages, and the rest of the state identified by this structure **must** adhere to the pipeline linking rules described in the Shader Interfaces chapter
- If rasterization is not disabled and subpass uses a depth/stencil attachment in renderpass that has a layout of VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL in the VkAttachmentReference defined by subpass, the depthWriteEnable member of pDepthStencilState **must** be VK_FALSE
- If rasterization is not disabled and subpass uses a depth/stencil attachment in renderpass that has a layout of VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL in the VkAttachmentReference defined by subpass, the failOp, passOp and depthFailOp members of each of the front and back members of pDepthStencilState **must** be VK_STENCIL_OP_KEEP
- If rasterization is not disabled and the subpass uses color attachments, then for each color attachment in the subpass the blendEnable member of the corresponding element of the pAttachment member of pColorBlendState **must** be VK_FALSE if the format of the attachment does not support color blend operations, as specified by the VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT flag in VkFormatProperties::linearTilingFeatures or VkFormatProperties::optimalTilingFeatures returned by vkGetPhysicalDeviceFormatProperties
- If rasterization is not disabled and the subpass uses color attachments, the attachmentCount member of pColorBlendState **must** be equal to the colorAttachmentCount used to create subpass
- If no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_VIEWPORT, the pViewports member of pViewportState **must** be a pointer to an array of pViewportState::viewportCount VkViewport structures
- If no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_SCISSOR, the pScissors member of pViewportState **must** be a pointer to an array of pViewportState::scissorCount VkRect2D structures
- If the wide lines 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
- If the rasterizerDiscardEnable member of pRasterizationState is VK_FALSE, pViewportState **must** be a pointer to a valid VkPipelineViewportStateCreateInfo structure
- If the rasterizerDiscardEnable member of pRasterizationState is VK_FALSE, pMultisampleState **must** be a pointer to a valid VkPipelineMultisampleStateCreateInfo

structure

- If the rasterizerDiscardEnable member of pRasterizationState is VK_FALSE, and subpass uses a depth/stencil attachment, pDepthStencilState **must** be a pointer to a valid VkPipelineDepthStencilStateCreateInfo structure
- If the rasterizerDiscardEnable member of pRasterizationState is VK_FALSE, and subpass uses color attachments, pColorBlendState **must** be a pointer to a valid VkPipelineColorBlendStateCreateInfo structure
- If the depth bias clamping feature is not enabled, no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_DEPTH_BIAS, and the depthBiasEnable member of pDepthStencil is VK_TRUE, the depthBiasClamp member of pDepthStencil **must** be 0.0
- If no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_DEPTH_BOUNDS, and the depthBoundsTestEnable member of pDepthStencil is VK_TRUE, the minDepthBounds and maxDepthBounds members of pDepthStencil **must** be between 0.0 and 1.0, inclusive
- layout **must** be consistent with all shaders specified in pStages
- If subpass uses color and/or depth/stencil attachments, then the rasterizationSamples member of pMultisampleState **must** be the same as the sample count for those subpass attachments
- If 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
- subpass **must** be a valid subpass within renderpass

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be a valid combination of VkPipelineCreateFlagBits values
- `pStages` **must** be a pointer to an array of `stageCount` valid `VkPipelineShaderStageCreateInfo` structures
- `pVertexInputState` **must** be a pointer to a valid `VkPipelineVertexInputStateCreateInfo` structure
- `pInputAssemblyState` **must** be a pointer to a valid `VkPipelineInputAssemblyStateCreateInfo` structure
- `pRasterizationState` **must** be a pointer to a valid `VkPipelineRasterizationStateCreateInfo` structure
- If `pDynamicState` is not `NULL`, `pDynamicState` **must** be a pointer to a valid `VkPipelineDynamicStateCreateInfo` structure
- `layout` **must** be a valid `VkPipelineLayout` handle
- `renderPass` **must** be a valid `VkRenderPass` handle
- `stageCount` **must** be greater than `0`
- Each of `basePipelineHandle`, `layout`, and `renderPass` that are valid handles **must** have been created, allocated, or retrieved from the same `VkDevice`

See Also

VkPipeline, VkPipelineColorBlendStateCreateInfo, VkPipelineCreateFlags, VkPipelineDepthStencilStateCreateInfo, VkPipelineDynamicStateCreateInfo, VkPipelineInputAssemblyStateCreateInfo, VkPipelineLayout, VkPipelineMultisampleStateCreateInfo, VkPipelineRasterizationStateCreateInfo, VkPipelineShaderStageCreateInfo, VkPipelineTessellationStateCreateInfo, VkPipelineVertexInputStateCreateInfo, VkPipelineViewportStateCreateInfo, VkRenderPass, VkStructureType, vkCreateGraphicsPipelines

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkGraphicsPipelineCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageBlit(3)

Name

VkImageBlit - Structure specifying an image blit operation

C Specification

The `VkImageBlit` structure is defined as:

```
typedef struct VkImageBlit {
    VkImageSubresourceLayers    srcSubresource;
    VkOffset3D                  srcOffsets[2];
    VkImageSubresourceLayers    dstSubresource;
    VkOffset3D                  dstOffsets[2];
} VkImageBlit;
```

Members

- `srcSubresource` is the subresource to blit from.
- `srcOffsets` is an array of two VkOffset3D structures specifying the bounds of the source region within `srcSubresource`.
- `dstSubresource` is the subresource to blit into.
- `dstOffsets` is an array of two VkOffset3D structures specifying the bounds of the destination region within `dstSubresource`.

Description

For each element of the `pRegions` array, a blit operation is performed the specified source and destination regions.

Valid Usage

- The `aspectMask` member of `srcSubresource` and `dstSubresource` **must** match
- The `layerCount` member of `srcSubresource` and `dstSubresource` **must** match
- If either of the calling command's `srcImage` or `dstImage` parameters are of VkImageType `VK_IMAGE_TYPE_3D`, the `baseArrayLayer` and `layerCount` members of both `srcSubresource` and `dstSubresource` **must** be `0` and `1`, respectively
- The `aspectMask` member of `srcSubresource` **must** specify aspects present in the calling command's `srcImage`
- The `aspectMask` member of `dstSubresource` **must** specify aspects present in the calling command's `dstImage`
- `srcOffset`[0].`x` and `srcOffset`[1].`x` **must** both be greater than or equal to `0` and less than or equal to the source image subresource width
- `srcOffset`[0].`y` and `srcOffset`[1].`y` **must** both be greater than or equal to `0` and less than or equal to the source image subresource height
- If the calling command's `srcImage` is of type `VK_IMAGE_TYPE_1D`, then `srcOffset`[0].y **must** be `0` and `srcOffset`[1].y **must** be `1`.
- `srcOffset`[0].`z` and `srcOffset`[1].`z` **must** both be greater than or equal to `0` and less than or equal to the source image subresource depth
- If the calling command's `srcImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then `srcOffset`[0].z **must** be `0` and `srcOffset`[1].z **must** be `1`.
- `dstOffset`[0].`x` and `dstOffset`[1].`x` **must** both be greater than or equal to `0` and less than or equal to the destination image subresource width
- `dstOffset`[0].`y` and `dstOffset`[1].`y` **must** both be greater than or equal to `0` and less than or equal to the destination image subresource height
- If the calling command's `dstImage` is of type `VK_IMAGE_TYPE_1D`, then `dstOffset`[0].y **must** be `0` and `dstOffset`[1].y **must** be `1`.
- `dstOffset`[0].`z` and `dstOffset`[1].`z` **must** both be greater than or equal to `0` and less than or equal to the destination image subresource depth
- If the calling command's `dstImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then `dstOffset`[0].z **must** be `0` and `dstOffset`[1].z **must** be `1`.

Valid Usage (Implicit)

- `srcSubresource` **must** be a valid `VkImageSubresourceLayers` structure
- `dstSubresource` **must** be a valid `VkImageSubresourceLayers` structure

See Also

VkImageSubresourceLayers, VkOffset3D, vkCmdBlitImage

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageBlit

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageCopy(3)

Name

VkImageCopy - Structure specifying an image copy operation

C Specification

The `VkImageCopy` structure is defined as:

```
typedef struct VkImageCopy {
    VkImageSubresourceLayers    srcSubresource;
    VkOffset3D                  srcOffset;
    VkImageSubresourceLayers    dstSubresource;
    VkOffset3D                  dstOffset;
    VkExtent3D                  extent;
} VkImageCopy;
```

Members

- `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 source image to copy in `width`, `height` and `depth`.

Description

Copies are done layer by layer starting with `baseArrayLayer` member of `srcSubresource` for the source and `dstSubresource` for the destination. `layerCount` layers are copied to the destination image.

Valid Usage

- The `aspectMask` member of `srcSubresource` and `dstSubresource` **must** match
- The `layerCount` member of `srcSubresource` and `dstSubresource` **must** match
- If either of the calling command's `srcImage` or `dstImage` parameters are of VkImageType `VK_IMAGE_TYPE_3D`, the `baseArrayLayer` and `layerCount` members of both `srcSubresource` and `dstSubresource` **must** be `0` and `1`, respectively
- The `aspectMask` member of `srcSubresource` **must** specify aspects present in the calling command's `srcImage`
- The `aspectMask` member of `dstSubresource` **must** specify aspects present in the calling command's `dstImage`
- `srcOffset.x` and (`extent.width` + `srcOffset.x`) **must** both be greater than or equal to `0` and less than or equal to the source image subresource width
- `srcOffset.y` and (`extent.height` + `srcOffset.y`) **must** both be greater than or equal to `0` and less than or equal to the source image subresource height
- If the calling command's `srcImage` is of type `VK_IMAGE_TYPE_1D`, then `srcOffset.y` **must** be `0` and `extent.height` **must** be `1`.
- `srcOffset.z` and (`extent.depth` + `srcOffset.z`) **must** both be greater than or equal to `0` and less than or equal to the source image subresource depth
- If the calling command's `srcImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then `srcOffset.z` **must** be `0` and `extent.depth` **must** be `1`.
- `srcSubresource.baseArrayLayer` **must** be less than and (`srcSubresource.layerCount` + `srcSubresource.baseArrayLayer`) **must** be less than or equal to the number of layers in the source image
- `dstOffset.x` and (`extent.width` + `dstOffset.x`) **must** both be greater than or equal to `0` and less than or equal to the destination image subresource width
- `dstOffset.y` and (`extent.height` + `dstOffset.y`) **must** both be greater than or equal to `0` and less than or equal to the destination image subresource height
- If the calling command's `dstImage` is of type `VK_IMAGE_TYPE_1D`, then `dstOffset.y` **must** be `0` and `extent.height` **must** be `1`.
- `dstOffset.z` and (`extent.depth` + `dstOffset.z`) **must** both be greater than or equal to `0` and less than or equal to the destination image subresource depth
- If the calling command's `dstImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then `dstOffset.z` **must** be `0` and `extent.depth` **must** be `1`.
- `dstSubresource.baseArrayLayer` **must** be less than and (`dstSubresource.layerCount` + `dstSubresource.baseArrayLayer`) **must** be less than or equal to the number of layers in the destination image
- If the calling command's `srcImage` is a compressed format image, all members of `srcOffset` **must** be a multiple of the corresponding dimensions of the compressed texel block
- If the calling command's `srcImage` is a compressed format image, `extent.width` **must** be a

multiple of the compressed texel block width or (`extent.width` + `srcOffset.x`) **must** equal the source image subresource width

- If the calling command's `srcImage` is a compressed format image, `extent.height` **must** be a multiple of the compressed texel block height or (`extent.height` + `srcOffset.y`) **must** equal the source image subresource height
- If the calling command's `srcImage` is a compressed format image, `extent.depth` **must** be a multiple of the compressed texel block depth or (`extent.depth` + `srcOffset.z`) **must** equal the source image subresource depth
- If the calling command's `dstImage` is a compressed format image, all members of `dstOffset` **must** be a multiple of the corresponding dimensions of the compressed texel block
- If the calling command's `dstImage` is a compressed format image, `extent.width` **must** be a multiple of the compressed texel block width or (`extent.width` + `dstOffset.x`) **must** equal the destination image subresource width
- If the calling command's `dstImage` is a compressed format image, `extent.height` **must** be a multiple of the compressed texel block height or (`extent.height` + `dstOffset.y`) **must** equal the destination image subresource height
- If the calling command's `dstImage` is a compressed format image, `extent.depth` **must** be a multiple of the compressed texel block depth or (`extent.depth` + `dstOffset.z`) **must** equal the destination image subresource depth
- `srcOffset`, `dstOffset`, and `extent` **must** respect the image transfer granularity requirements of the queue family that it will be submitted against, as described in Physical Device Enumeration

Valid Usage (Implicit)

- `srcSubresource` **must** be a valid `VkImageSubresourceLayers` structure
- `dstSubresource` **must** be a valid `VkImageSubresourceLayers` structure

See Also

VkExtent3D, VkImageSubresourceLayers, VkOffset3D, vkCmdCopyImage

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageCopy

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageCreateInfo(3)

Name

VkImageCreateInfo - Structure specifying the parameters of a newly created image object

C Specification

The `VkImageCreateInfo` structure is defined as:

```
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;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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 data elements that will be contained in the image.
- `extent` is a VkExtent3D describing the number of data elements 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 the number of sub-data element samples in the image as defined in VkSampleCountFlagBits. See Multisampling.
- `tiling` is a VkImageTiling value specifying the tiling arrangement of the data elements 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 list of queue families that will access this image (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.

Description

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`

Implementations **may** support additional limits and capabilities beyond those listed above.

To query an implementation's specific capabilities for a given combination of `format`, `imageType`, `tiling`, `usage`, and `flags`, call vkGetPhysicalDeviceImageFormatProperties. The return value indicates whether that combination of image settings is supported. On success, the `VkImageFormatProperties` output parameter indicates the set of valid `samples` bits and the limits for `extent`, `mipLevels`, and `arrayLayers`.

To determine the set of valid `usage` bits for a given format, call vkGetPhysicalDeviceFormatProperties.

Valid Usage

- The combination of format, imageType, tiling, usage, and flags **must** be supported, as indicated by a VK_SUCCESS return value from vkGetPhysicalDeviceImageFormatProperties invoked with the same values passed to the corresponding parameters.
- If sharingMode is VK_SHARING_MODE_CONCURRENT, pQueueFamilyIndices **must** be a pointer to an array of queueFamilyIndexCount uint32_t values
- If sharingMode is VK_SHARING_MODE_CONCURRENT, queueFamilyIndexCount **must** be greater than 1
- If sharingMode is VK_SHARING_MODE_CONCURRENT, each element of pQueueFamilyIndices **must** be unique and **must** be less than pQueueFamilyPropertyCount returned by vkGetPhysicalDeviceQueueFamilyProperties for the physicalDevice that was used to create device
- format **must** not be VK_FORMAT_UNDEFINED
- extent::width **must** be greater than 0.
- extent::height **must** be greater than 0.
- extent::depth **must** be greater than 0.
- mipLevels **must** be greater than 0
- arrayLayers **must** be greater than 0
- If flags contains VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT, imageType **must** be VK_IMAGE_TYPE_2D
- If imageType is VK_IMAGE_TYPE_1D, extent.width **must** be less than or equal to VkPhysicalDeviceLimits::maxImageDimension1D, or VkImageFormatProperties::maxExtent.width (as returned by vkGetPhysicalDeviceImageFormatProperties with format, imageType, tiling, usage, and flags equal to those in this structure) - whichever is higher
- If imageType is VK_IMAGE_TYPE_2D and flags does not contain VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT, extent.width and extent.height **must** be less than or equal to VkPhysicalDeviceLimits::maxImageDimension2D, or VkImageFormatProperties ::maxExtent.width/height (as returned by vkGetPhysicalDeviceImageFormatProperties with format, imageType, tiling, usage, and flags equal to those in this structure) - whichever is higher
- If imageType is VK_IMAGE_TYPE_2D and flags contains VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT, extent.width and extent.height **must** be less than or equal to VkPhysicalDeviceLimits ::maxImageDimensionCube, or VkImageFormatProperties::maxExtent.width/height (as returned by vkGetPhysicalDeviceImageFormatProperties with format, imageType, tiling, usage, and flags equal to those in this structure) - whichever is higher
- If imageType is VK_IMAGE_TYPE_2D and flags contains VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT, extent.width and extent.height **must** be equal and arrayLayers **must** be greater than or equal to 6
- If imageType is VK_IMAGE_TYPE_3D, extent.width, extent.height and extent.depth **must** be less than or equal to VkPhysicalDeviceLimits::maxImageDimension3D, or VkImageFormatProperties ::maxExtent.width/height/depth (as returned by vkGetPhysicalDeviceImageFormatProperties

with format, imageType, tiling, usage, and flags equal to those in this structure) - whichever is higher

- If imageType is VK_IMAGE_TYPE_1D, both extent.height and extent.depth **must** be 1
- If imageType is VK_IMAGE_TYPE_2D, extent.depth **must** be 1
- mipLevels **must** be less than or equal to $\lfloor\log_2(\max($extent.width, extent.height, extent.depth$))\rfloor + 1$.
- If any of extent.width, extent.height, or extent.depth are greater than the equivalently named members of VkPhysicalDeviceLimits::maxImageDimension3D, mipLevels **must** be less than or equal to VkImageFormatProperties::maxMipLevels (as returned by vkGetPhysicalDeviceImageFormatProperties with format, imageType, tiling, usage, and flags equal to those in this structure)
- arrayLayers **must** be less than or equal to VkImageFormatProperties::maxArrayLayers (as returned by vkGetPhysicalDeviceImageFormatProperties with format, imageType, tiling, usage, and flags equal to those in this structure)
- If imageType is VK_IMAGE_TYPE_3D, arrayLayers **must** be 1.
- If samples is not VK_SAMPLE_COUNT_1_BIT, imageType **must** be VK_IMAGE_TYPE_2D, flags **must** not contain VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT, tiling **must** be VK_IMAGE_TILING_OPTIMAL, and mipLevels **must** be equal to 1
- 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
- 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
- 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
- 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.
- samples **must** be a bit value that is set in VkImageFormatProperties::sampleCounts returned by vkGetPhysicalDeviceImageFormatProperties with format, imageType, tiling, usage, and flags equal to those in this structure
- If the multisampled storage images feature is not enabled, and usage contains VK_IMAGE_USAGE_STORAGE_BIT, samples **must** be VK_SAMPLE_COUNT_1_BIT
- If the sparse bindings feature is not enabled, flags **must** not contain VK_IMAGE_CREATE_SPARSE_BINDING_BIT
- If imageType is VK_IMAGE_TYPE_1D, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- If the sparse residency for 2D images feature is not enabled, and imageType is

VK_IMAGE_TYPE_2D, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT

- If the sparse residency for 3D images feature is not enabled, and imageType is VK_IMAGE_TYPE_3D, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- If the sparse residency for images with 2 samples feature is not enabled, imageType is VK_IMAGE_TYPE_2D, and samples is VK_SAMPLE_COUNT_2_BIT, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- If the sparse residency for images with 4 samples feature is not enabled, imageType is VK_IMAGE_TYPE_2D, and samples is VK_SAMPLE_COUNT_4_BIT, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- If the sparse residency for images with 8 samples feature is not enabled, imageType is VK_IMAGE_TYPE_2D, and samples is VK_SAMPLE_COUNT_8_BIT, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- If the sparse residency for images with 16 samples feature is not enabled, imageType is VK_IMAGE_TYPE_2D, and samples is VK_SAMPLE_COUNT_16_BIT, flags **must** not contain VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT
- If tiling is VK_IMAGE_TILING_LINEAR, format **must** be a format that has at least one supported feature bit present in the value of VkFormatProperties::linearTilingFeatures returned by vkGetPhysicalDeviceFormatProperties with the same value of format
- If tiling is VK_IMAGE_TILING_LINEAR, and VkFormatProperties::linearTilingFeatures (as returned by vkGetPhysicalDeviceFormatProperties with the same value of format) does not include VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT, usage **must** not contain VK_IMAGE_USAGE_SAMPLED_BIT
- If tiling is VK_IMAGE_TILING_LINEAR, and VkFormatProperties::linearTilingFeatures (as returned by vkGetPhysicalDeviceFormatProperties with the same value of format) does not include VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT, usage **must** not contain VK_IMAGE_USAGE_STORAGE_BIT
- If tiling is VK_IMAGE_TILING_LINEAR, and VkFormatProperties::linearTilingFeatures (as returned by vkGetPhysicalDeviceFormatProperties with the same value of format) does not include VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT, usage **must** not contain VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
- If tiling is VK_IMAGE_TILING_LINEAR, and VkFormatProperties::linearTilingFeatures (as returned by vkGetPhysicalDeviceFormatProperties with the same value of format) does not include VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT, usage **must** not contain VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- If tiling is VK_IMAGE_TILING_OPTIMAL, format **must** be a format that has at least one supported feature bit present in the value of VkFormatProperties::optimalTilingFeatures returned by vkGetPhysicalDeviceFormatProperties with the same value of format
- If tiling is VK_IMAGE_TILING_OPTIMAL, and VkFormatProperties::optimalTilingFeatures (as returned by vkGetPhysicalDeviceFormatProperties with the same value of format) does not include VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT, usage **must** not contain VK_IMAGE_USAGE_SAMPLED_BIT
- If tiling is VK_IMAGE_TILING_OPTIMAL, and VkFormatProperties::optimalTilingFeatures (as returned by vkGetPhysicalDeviceFormatProperties with the same value of format) does not include VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT, usage **must** not contain VK_IMAGE_USAGE_STORAGE_BIT

- If `tiling` is `VK_IMAGE_TILING_OPTIMAL`, and `VkFormatProperties`::`optimalTilingFeatures` (as returned by `vkGetPhysicalDeviceFormatProperties` with the same value of `format`) does not include `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`, `usage` **must** not contain `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
- If `tiling` is `VK_IMAGE_TILING_OPTIMAL`, and `VkFormatProperties`::`optimalTilingFeatures` (as returned by `vkGetPhysicalDeviceFormatProperties` with the same value of `format`) does not include `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT`, `usage` **must** not contain `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- If `flags` contains `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` or `VK_IMAGE_CREATE_SPARSE_ALIASED_BIT`, it **must** also contain `VK_IMAGE_CREATE_SPARSE_BINDING_BIT`
- `initialLayout` **must** be `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`.

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be a valid combination of VkImageCreateFlagBits values
- `imageType` **must** be a valid VkImageType value
- `format` **must** be a valid VkFormat value
- `samples` **must** be a valid VkSampleCountFlagBits value
- `tiling` **must** be a valid VkImageTiling value
- `usage` **must** be a valid combination of VkImageUsageFlagBits values
- `usage` **must** not be `0`
- `sharingMode` **must** be a valid VkSharingMode value
- `initialLayout` **must** be a valid VkImageLayout value

See Also

VkExtent3D, VkFormat, VkImageCreateFlags, VkImageLayout, VkImageTiling, VkImageType, VkImageUsageFlags, VkSampleCountFlagBits, VkSharingMode, VkStructureType, vkCreateImage

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageFormatProperties(3)

Name

VkImageFormatProperties - Structure specifying a image format properties

C Specification

The `VkImageFormatProperties` structure is defined as:

```
typedef struct VkImageFormatProperties {
    VkExtent3D            maxExtent;
    uint32_t              maxMipLevels;
    uint32_t              maxArrayLayers;
    VkSampleCountFlags    sampleCounts;
    VkDeviceSize          maxResourceSize;
} VkImageFormatProperties;
```

Members

- `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** either be equal to 1 (valid only if `tiling` is `VK_IMAGE_TILING_LINEAR`) or be equal to ⌈log$_2$(max(`width`, `height`, `depth`))⌉ + 1. `width`, `height`, and `depth` are taken from the corresponding members of `maxExtent`.
- `maxArrayLayers` is the maximum number of array layers. `maxArrayLayers` **must** either be equal to 1 or be greater than or equal to the `maxImageArrayLayers` member of VkPhysicalDeviceLimits. A value of 1 is valid only if `tiling` is `VK_IMAGE_TILING_LINEAR` or if `type` is `VK_IMAGE_TYPE_3D`.
- `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}.

Description

> *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 or the image will be invalid. 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.

See Also

`VkDeviceSize`, VkExtent3D, VkSampleCountFlags, vkGetPhysicalDeviceImageFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageFormatProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageMemoryBarrier(3)

Name

VkImageMemoryBarrier - Structure specifying the parameters of an image memory barrier

C Specification

The `VkImageMemoryBarrier` structure is defined as:

```
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;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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.

Description

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 access scope includes no access, as if dstAccessMask was 0.

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 access scope includes no access, as if srcAccessMask was 0.

If oldLayout is not equal to newLayout, then the memory barrier defines an image layout transition for the specified image subresource range.

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. In effect there is an implicit execution dependency from each such layout transition to all layout transitions previously submitted to the same queue.

Valid Usage

- `oldLayout` **must** be `VK_IMAGE_LAYOUT_UNDEFINED` or the current layout of the image subresources affected by the barrier
- `newLayout` **must** not be `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`
- If `image` was created with a sharing mode of `VK_SHARING_MODE_CONCURRENT`, `srcQueueFamilyIndex` and `dstQueueFamilyIndex` **must** both be `VK_QUEUE_FAMILY_IGNORED`
- If `image` was created with a sharing mode of `VK_SHARING_MODE_EXCLUSIVE`, `srcQueueFamilyIndex` and `dstQueueFamilyIndex` **must** either both be `VK_QUEUE_FAMILY_IGNORED`, or both be a valid queue family (see html/vkspec.html#devsandqueues-queueprops).
- If `image` was created with a sharing mode of `VK_SHARING_MODE_EXCLUSIVE`, and `srcQueueFamilyIndex` and `dstQueueFamilyIndex` are not `VK_QUEUE_FAMILY_IGNORED`, at least one of them **must** be the same as the family of the queue that will execute this barrier
- `subresourceRange`::`baseMipLevel` **must** be less than the `mipLevels` specified in VkImageCreateInfo when `image` was created
- If `subresourceRange`::`levelCount` is not `VK_REMAINING_MIP_LEVELS`, `subresourceRange`::`levelCount` **must** be non-zero and `subresourceRange`::`baseMipLevel` + `subresourceRange`::`levelCount` **must** be less than or equal to the `mipLevels` specified in VkImageCreateInfo when `image` was created
- `subresourceRange`::`baseArrayLayer` **must** be less than the `arrayLayers` specified in VkImageCreateInfo when `image` was created
- If `subresourceRange`::`layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `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
- If `image` has a depth/stencil format with both depth and stencil components, then `aspectMask` member of `subresourceRange` **must** include both `VK_IMAGE_ASPECT_DEPTH_BIT` and `VK_IMAGE_ASPECT_STENCIL_BIT`
- If either `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` set
- If either `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` set
- If either `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` set
- If either `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` set
- If either `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_SRC_BIT` set
- If either `oldLayout` or `newLayout` is `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL` then `image` **must** have been created with `VK_IMAGE_USAGE_TRANSFER_DST_BIT` set

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER`
- `pNext` **must** be `NULL`
- `srcAccessMask` **must** be a valid combination of VkAccessFlagBits values
- `dstAccessMask` **must** be a valid combination of VkAccessFlagBits values
- `oldLayout` **must** be a valid VkImageLayout value
- `newLayout` **must** be a valid VkImageLayout value
- `image` **must** be a valid `VkImage` handle
- `subresourceRange` **must** be a valid `VkImageSubresourceRange` structure

See Also

VkAccessFlags, VkImage, VkImageLayout, VkImageSubresourceRange, VkStructureType, vkCmdPipelineBarrier, vkCmdWaitEvents

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageMemoryBarrier

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageResolve(3)

Name

VkImageResolve - Structure specifying an image resolve operation

C Specification

The `VkImageResolve` structure is defined as:

```
typedef struct VkImageResolve {
    VkImageSubresourceLayers    srcSubresource;
    VkOffset3D                  srcOffset;
    VkImageSubresourceLayers    dstSubresource;
    VkOffset3D                  dstOffset;
    VkExtent3D                  extent;
} VkImageResolve;
```

Members

- `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`.

Description

Valid Usage

- The `aspectMask` member of `srcSubresource` and `dstSubresource` **must** only contain `VK_IMAGE_ASPECT_COLOR_BIT`
- The `layerCount` member of `srcSubresource` and `dstSubresource` **must** match
- If either of the calling command's `srcImage` or `dstImage` parameters are of VkImageType `VK_IMAGE_TYPE_3D`, the `baseArrayLayer` and `layerCount` members of both `srcSubresource` and `dstSubresource` **must** be `0` and `1`, respectively
- `srcOffset.x` and (`extent.width` + `srcOffset.x`) **must** both be greater than or equal to `0` and less than or equal to the source image subresource width
- `srcOffset.y` and (`extent.height` + `srcOffset.y`) **must** both be greater than or equal to `0` and less than or equal to the source image subresource height
- If the calling command's `srcImage` is of type `VK_IMAGE_TYPE_1D`, then `srcOffset.y` **must** be `0` and `extent.height` **must** be `1`.
- `srcOffset.z` and (`extent.depth` + `srcOffset.z`) **must** both be greater than or equal to `0` and less than or equal to the source image subresource depth
- If the calling command's `srcImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then `srcOffset.z` **must** be `0` and `extent.depth` **must** be `1`.
- `dstOffset.x` and (`extent.width` + `dstOffset.x`) **must** both be greater than or equal to `0` and less than or equal to the destination image subresource width
- `dstOffset.y` and (`extent.height` + `dstOffset.y`) **must** both be greater than or equal to `0` and less than or equal to the destination image subresource height
- If the calling command's `dstImage` is of type `VK_IMAGE_TYPE_1D`, then `dstOffset.y` **must** be `0` and `extent.height` **must** be `1`.
- `dstOffset.z` and (`extent.depth` + `dstOffset.z`) **must** both be greater than or equal to `0` and less than or equal to the destination image subresource depth
- If the calling command's `dstImage` is of type `VK_IMAGE_TYPE_1D` or `VK_IMAGE_TYPE_2D`, then `dstOffset.z` **must** be `0` and `extent.depth` **must** be `1`.

Valid Usage (Implicit)

- `srcSubresource` **must** be a valid `VkImageSubresourceLayers` structure
- `dstSubresource` **must** be a valid `VkImageSubresourceLayers` structure

See Also

VkExtent3D, VkImageSubresourceLayers, VkOffset3D, vkCmdResolveImage

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageResolve

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageSubresource(3)

Name

VkImageSubresource - Structure specifying a image subresource

C Specification

The `VkImageSubresource` structure is defined as:

```
typedef struct VkImageSubresource {
    VkImageAspectFlags    aspectMask;
    uint32_t              mipLevel;
    uint32_t              arrayLayer;
} VkImageSubresource;
```

Members

- `aspectMask` is a VkImageAspectFlags selecting the image *aspect*.
- `mipLevel` selects the mipmap level.
- `arrayLayer` selects the array layer.

Description

Valid Usage

- `mipLevel` **must** be less than the `mipLevels` specified in VkImageCreateInfo when the image was created
- `arrayLayer` **must** be less than the `arrayLayers` specified in VkImageCreateInfo when the image was created

Valid Usage (Implicit)

- `aspectMask` **must** be a valid combination of VkImageAspectFlagBits values
- `aspectMask` **must** not be `0`

See Also

VkImageAspectFlags, VkSparseImageMemoryBind, vkGetImageSubresourceLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageSubresource

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageSubresourceLayers(3)

Name

VkImageSubresourceLayers - Structure specifying a image subresource layers

C Specification

The `VkImageSubresourceLayers` structure is defined as:

```
typedef struct VkImageSubresourceLayers {
    VkImageAspectFlags    aspectMask;
    uint32_t              mipLevel;
    uint32_t              baseArrayLayer;
    uint32_t              layerCount;
} VkImageSubresourceLayers;
```

Members

- `aspectMask` is a combination of VkImageAspectFlagBits, selecting the color, depth and/or stencil aspects to be copied.
- `mipLevel` is the mipmap level to copy from.
- `baseArrayLayer` and `layerCount` are the starting layer and number of layers to copy.

Description

Valid Usage

- 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`
- `aspectMask` **must** not contain `VK_IMAGE_ASPECT_METADATA_BIT`
- `mipLevel` **must** be less than the `mipLevels` specified in VkImageCreateInfo when the image was created
- (`baseArrayLayer + layerCount`) **must** be less than or equal to the `arrayLayers` specified in VkImageCreateInfo when the image was created

Valid Usage (Implicit)

- `aspectMask` **must** be a valid combination of VkImageAspectFlagBits values
- `aspectMask` **must** not be `0`

See Also

VkBufferImageCopy, VkImageAspectFlags, VkImageBlit, VkImageCopy, VkImageResolve

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageSubresourceLayers

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageSubresourceRange(3)

Name

VkImageSubresourceRange - Structure specifying a image subresource range

C Specification

The `VkImageSubresourceRange` structure is defined as:

```
typedef struct VkImageSubresourceRange {
    VkImageAspectFlags    aspectMask;
    uint32_t              baseMipLevel;
    uint32_t              levelCount;
    uint32_t              baseArrayLayer;
    uint32_t              layerCount;
} VkImageSubresourceRange;
```

Members

- `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.

Description

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, behavior when indexing the last cube is undefined.

`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. 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 using an imageView 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 and selects whether the imageView 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 imageView 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.

The `components` member is of type VkComponentMapping, and describes a remapping from components of the image to components of the vector returned by shader image instructions. This remapping **must** be identity for storage image descriptors, input attachment descriptors, and framebuffer attachments.

Valid Usage (Implicit)

- `aspectMask` **must** be a valid combination of VkImageAspectFlagBits values
- `aspectMask` **must** not be `0`

See Also

VkImageAspectFlags, VkImageMemoryBarrier, VkImageViewCreateInfo, vkCmdClearColorImage, vkCmdClearDepthStencilImage

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageSubresourceRange

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageViewCreateInfo(3)

Name

VkImageViewCreateInfo - Structure specifying parameters of a newly created image view

C Specification

The `VkImageViewCreateInfo` structure is defined as:

```
typedef struct VkImageViewCreateInfo {
    VkStructureType            sType;
    const void*                pNext;
    VkImageViewCreateFlags     flags;
    VkImage                    image;
    VkImageViewType            viewType;
    VkFormat                   format;
    VkComponentMapping         components;
    VkImageSubresourceRange    subresourceRange;
} VkImageViewCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `image` is a `VkImage` on which the view will be created.
- `viewType` is an VkImageViewType value specifying the type of the image view.
- `format` is a VkFormat describing the format and type used to interpret data elements in the image.
- `components` is a VkComponentMapping specifies a remapping of color components (or of depth or stencil components after they have been converted into color components).
- `subresourceRange` is a VkImageSubresourceRange selecting the set of mipmap levels and array layers to be accessible to the view.

Description

If `image` was created with the `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` flag, `format` **can** be different from the image's format, but if 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.

Table 6. Image and image view parameter compatibility requirements

Dim, Arrayed, MS	Image parameters	View parameters

	`imageType` = ci.`imageType`
	`width` = ci.`extent.width`
	`height` = ci.`extent.height`
	`depth` = ci.`extent.depth`
	`arrayLayers` = ci.`arrayLayers`
	`samples` = ci.`samples`
	`flags` = ci.`flags`
	where ci is the VkImageCreateInfo used to create `image`.
		`baseArrayLayer` and `layerCount` are members of the `subresourceRange` member.

1D, 0, 0	`imageType` = `VK_IMAGE_TYPE_1D` `width` ≥ 1 `height` = 1 `depth` = 1 `arrayLayers` ≥ 1 `samples` = 1	`viewType` = `VK_IMAGE_VIEW_TYPE_1D` `baseArrayLayer` ≥ 0 `layerCount` = 1
1D, 1, 0	`imageType` = `VK_IMAGE_TYPE_1D` `width` ≥ 1 `height` = 1 `depth` = 1 `arrayLayers` ≥ 1 `samples` = 1	`viewType` = `VK_IMAGE_VIEW_TYPE_1D_ARRAY` `baseArrayLayer` ≥ 0 `layerCount` ≥ 1
2D, 0, 0	`imageType` = `VK_IMAGE_TYPE_2D` `width` ≥ 1 `height` ≥ 1 `depth` = 1 `arrayLayers` ≥ 1 `samples` = 1	`viewType` = `VK_IMAGE_VIEW_TYPE_2D` `baseArrayLayer` ≥ 0 `layerCount` = 1
2D, 1, 0	`imageType` = `VK_IMAGE_TYPE_2D` `width` ≥ 1 `height` ≥ 1 `depth` = 1 `arrayLayers` ≥ 1 `samples` = 1	`viewType` = `VK_IMAGE_VIEW_TYPE_2D_ARRAY` `baseArrayLayer` ≥ 0 `layerCount` ≥ 1
2D, 0, 1	`imageType` = `VK_IMAGE_TYPE_2D` `width` ≥ 1 `height` ≥ 1 `depth` = 1 `arrayLayers` ≥ 1 `samples` > 1	`viewType` = `VK_IMAGE_VIEW_TYPE_2D` `baseArrayLayer` ≥ 0 `layerCount` = 1
2D, 1, 1	`imageType` = `VK_IMAGE_TYPE_2D` `width` ≥ 1 `height` ≥ 1 `depth` = 1 `arrayLayers` ≥ 1 `samples` > 1	`viewType` = `VK_IMAGE_VIEW_TYPE_2D_ARRAY` `baseArrayLayer` ≥ 0 `layerCount` ≥ 1

Dim, Arrayed, MS	Image parameters	View parameters

CUBE, 0, 0	`imageType` = `VK_IMAGE_TYPE_2D`
	`width` ≥ 1
	`height` = `width`
	`depth` = 1
	`arrayLayers` ≥ 6
	`samples` = 1
	`flags` includes `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`
		`viewType` = `VK_IMAGE_VIEW_TYPE_CUBE`
		`baseArrayLayer` ≥ 0
		`layerCount` = 6

CUBE, 1, 0	`imageType` = `VK_IMAGE_TYPE_2D`
	`width` ≥ 1
	`height` = width
	`depth` = 1
	$N \geq 1$
	`arrayLayers` ≥ $6 \times N$
	`samples` = 1
	`flags` includes `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`
		`viewType` = `VK_IMAGE_VIEW_TYPE_CUBE_ARRAY`
		`baseArrayLayer` ≥ 0
		`layerCount` = $6 \times N$, $N \geq 1$

3D, 0, 0	`imageType` = `VK_IMAGE_TYPE_3D` `width` ≥ 1 `height` ≥ 1 `depth` ≥ 1 `arrayLayers` = 1 `samples` = 1	`viewType` = `VK_IMAGE_VIEW_TYPE_3D` `baseArrayLayer` = 0 `layerCount` = 1

Valid Usage

- 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
- If the image cubemap arrays feature is not enabled, viewType **must** not be VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
- If image was created with VK_IMAGE_TILING_LINEAR, format **must** be format that has at least one supported feature bit present in the value of VkFormatProperties::linearTilingFeatures returned by vkGetPhysicalDeviceFormatProperties with the same value of format
- image **must** have been created with a usage value containing at least one of VK_IMAGE_USAGE_SAMPLED_BIT, VK_IMAGE_USAGE_STORAGE_BIT, VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT, VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
- If image was created with VK_IMAGE_TILING_LINEAR and usage contains VK_IMAGE_USAGE_SAMPLED_BIT, format **must** be supported for sampled images, as specified by the VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT flag in VkFormatProperties::linearTilingFeatures returned by vkGetPhysicalDeviceFormatProperties with the same value of format
- If image was created with VK_IMAGE_TILING_LINEAR and usage contains VK_IMAGE_USAGE_STORAGE_BIT, format **must** be supported for storage images, as specified by the VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT flag in VkFormatProperties::linearTilingFeatures returned by vkGetPhysicalDeviceFormatProperties with the same value of format
- If image was created with VK_IMAGE_TILING_LINEAR and usage contains VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT, format **must** be supported for color attachments, as specified by the VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT flag in VkFormatProperties ::linearTilingFeatures returned by vkGetPhysicalDeviceFormatProperties with the same value of format
- If image was created with VK_IMAGE_TILING_LINEAR and usage contains VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, format **must** be supported for depth/stencil attachments, as specified by the VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT flag in VkFormatProperties::linearTilingFeatures returned by vkGetPhysicalDeviceFormatProperties with the same value of format
- If image was created with VK_IMAGE_TILING_OPTIMAL, format **must** be format that has at least one supported feature bit present in the value of VkFormatProperties ::optimalTilingFeatures returned by vkGetPhysicalDeviceFormatProperties with the same value of format
- If image was created with VK_IMAGE_TILING_OPTIMAL and usage contains VK_IMAGE_USAGE_SAMPLED_BIT, format **must** be supported for sampled images, as specified by the VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT flag in VkFormatProperties::optimalTilingFeatures returned by vkGetPhysicalDeviceFormatProperties with the same value of format
- If image was created with VK_IMAGE_TILING_OPTIMAL and usage contains VK_IMAGE_USAGE_STORAGE_BIT, format **must** be supported for storage images, as specified by the VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT flag in VkFormatProperties::optimalTilingFeatures returned by vkGetPhysicalDeviceFormatProperties with the same value of format

- If image was created with VK_IMAGE_TILING_OPTIMAL and usage contains VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT, format **must** be supported for color attachments, as specified by the VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT flag in VkFormatProperties::optimalTilingFeatures returned by vkGetPhysicalDeviceFormatProperties with the same value of format
- If image was created with VK_IMAGE_TILING_OPTIMAL and usage contains VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, format **must** be supported for depth/stencil attachments, as specified by the VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT flag in VkFormatProperties::optimalTilingFeatures returned by vkGetPhysicalDeviceFormatProperties with the same value of format
- subresourceRange::baseMipLevel **must** be less than the mipLevels specified in VkImageCreateInfo when image was created
- If subresourceRange::levelCount is not VK_REMAINING_MIP_LEVELS, subresourceRange::levelCount **must** be non-zero and subresourceRange::baseMipLevel + subresourceRange::levelCount **must** be less than or equal to the mipLevels specified in VkImageCreateInfo when image was created
- subresourceRange::baseArrayLayer **must** be less than the arrayLayers specified in VkImageCreateInfo when image was created
- If subresourceRange::layerCount is not VK_REMAINING_ARRAY_LAYERS, 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
- If image was created with the VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT flag, format **must** be compatible with the format used to create image, as defined in Format Compatibility Classes
- If image was not created with the VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT flag, format **must** be identical to the format used to create image
- If image is non-sparse then it **must** be bound completely and contiguously to a single VkDeviceMemory object
- subresourceRange and viewType **must** be compatible with the image, as described in the compatibility table

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `image` **must** be a valid `VkImage` handle
- `viewType` **must** be a valid VkImageViewType value
- `format` **must** be a valid VkFormat value
- `components` **must** be a valid `VkComponentMapping` structure
- `subresourceRange` **must** be a valid `VkImageSubresourceRange` structure

See Also

VkComponentMapping, VkFormat, VkImage, VkImageSubresourceRange, VkImageViewCreateFlags, VkImageViewType, VkStructureType, vkCreateImageView

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageViewCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkInstanceCreateInfo(3)

Name

VkInstanceCreateInfo - Structure specifying parameters of a newly created instance

C Specification

The `VkInstanceCreateInfo` structure is defined as:

```
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;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `pApplicationInfo` is `NULL` or a pointer to an instance of `VkApplicationInfo`. 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. 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.

Description

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- If `pApplicationInfo` is not `NULL`, `pApplicationInfo` **must** be a pointer to a valid `VkApplicationInfo` structure
- If `enabledLayerCount` is not `0`, `ppEnabledLayerNames` **must** be a pointer to an array of `enabledLayerCount` null-terminated UTF-8 strings
- If `enabledExtensionCount` is not `0`, `ppEnabledExtensionNames` **must** be a pointer to an array of `enabledExtensionCount` null-terminated UTF-8 strings

See Also

VkApplicationInfo, VkInstanceCreateFlags, VkStructureType, vkCreateInstance

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkInstanceCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkLayerProperties(3)

Name

VkLayerProperties - Structure specifying layer properties

C Specification

The VkLayerProperties structure is defined as:

```
typedef struct VkLayerProperties {
    char        layerName[VK_MAX_EXTENSION_NAME_SIZE];
    uint32_t    specVersion;
    uint32_t    implementationVersion;
    char        description[VK_MAX_DESCRIPTION_SIZE];
} VkLayerProperties;
```

Members

- layerName is a null-terminated UTF-8 string specifying 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 the API Version Numbers and Semantics section.
- implementationVersion is the version of this layer. It is an integer, increasing with backward compatible changes.
- description is a null-terminated UTF-8 string providing additional details that **can** be used by the application to identify the layer.

Description

See Also

vkEnumerateDeviceLayerProperties, vkEnumerateInstanceLayerProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkLayerProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkMappedMemoryRange(3)

Name

VkMappedMemoryRange - Structure specifying a mapped memory range

C Specification

The VkMappedMemoryRange structure is defined as:

```
typedef struct VkMappedMemoryRange {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceMemory     memory;
    VkDeviceSize       offset;
    VkDeviceSize       size;
} VkMappedMemoryRange;
```

Members

- sType is the type of this structure.
- pNext is NULL or a pointer to an extension-specific 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.

Description

Valid Usage

- `memory` **must** currently be mapped
- If `size` is not equal to `VK_WHOLE_SIZE`, `offset` and `size` **must** specify a range contained within the currently mapped range of `memory`
- If `size` is equal to `VK_WHOLE_SIZE`, `offset` **must** be within the currently mapped range of `memory`
- If `size` is equal to `VK_WHOLE_SIZE`, the end of the current mapping of `memory` **must** be a multiple of `VkPhysicalDeviceLimits`::`nonCoherentAtomSize` bytes from the beginning of the memory object.
- `offset` **must** be a multiple of `VkPhysicalDeviceLimits`::`nonCoherentAtomSize`
- 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)

- `sType` **must** be `VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE`
- `pNext` **must** be `NULL`
- `memory` **must** be a valid `VkDeviceMemory` handle

See Also

VkDeviceMemory, `VkDeviceSize`, VkStructureType, vkFlushMappedMemoryRanges, vkInvalidateMappedMemoryRanges

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkMappedMemoryRange

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkMemoryAllocateInfo(3)

Name

VkMemoryAllocateInfo - Structure containing parameters of a memory allocation

C Specification

The `VkMemoryAllocateInfo` structure is defined as:

```
typedef struct VkMemoryAllocateInfo {
    VkStructureType    sType;
    const void*        pNext;
    VkDeviceSize       allocationSize;
    uint32_t           memoryTypeIndex;
} VkMemoryAllocateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `allocationSize` is the size of the allocation in bytes
- `memoryTypeIndex` is the memory type index, which selects the properties of the memory to be allocated, as well as the heap the memory will come from.

Description

Valid Usage

- `allocationSize` **must** be less than or equal to the amount of memory available to the `VkMemoryHeap` specified by `memoryTypeIndex` and the calling command's `VkDevice`
- `allocationSize` **must** be greater than `0`

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO`
- `pNext` **must** be `NULL`

See Also

`VkDeviceSize`, VkStructureType, vkAllocateMemory

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkMemoryAllocateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkMemoryBarrier(3)

Name

VkMemoryBarrier - Structure specifying a global memory barrier

C Specification

The `VkMemoryBarrier` structure is defined as:

```
typedef struct VkMemoryBarrier {
    VkStructureType    sType;
    const void*        pNext;
    VkAccessFlags      srcAccessMask;
    VkAccessFlags      dstAccessMask;
} VkMemoryBarrier;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `srcAccessMask` is a bitmask of VkAccessFlagBits specifying a source access mask.
- `dstAccessMask` is a bitmask of VkAccessFlagBits specifying a destination access mask.

Description

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)

- `sType` **must** be `VK_STRUCTURE_TYPE_MEMORY_BARRIER`
- `pNext` **must** be `NULL`
- `srcAccessMask` **must** be a valid combination of VkAccessFlagBits values
- `dstAccessMask` **must** be a valid combination of VkAccessFlagBits values

See Also

VkAccessFlags, VkStructureType, vkCmdPipelineBarrier, vkCmdWaitEvents

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkMemoryBarrier

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkMemoryHeap(3)

Name

VkMemoryHeap - Structure specifying a memory heap

C Specification

The `VkMemoryHeap` structure is defined as:

```
typedef struct VkMemoryHeap {
    VkDeviceSize         size;
    VkMemoryHeapFlags    flags;
} VkMemoryHeap;
```

Members

- `size` is the total memory size in bytes in the heap.
- `flags` is a bitmask of VkMemoryHeapFlagBits specifying attribute flags for the heap.

Description

See Also

`VkDeviceSize`, VkMemoryHeapFlags, VkPhysicalDeviceMemoryProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkMemoryHeap

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkMemoryRequirements(3)

Name

VkMemoryRequirements - Structure specifying memory requirements

C Specification

The `VkMemoryRequirements` structure is defined as:

```
typedef struct VkMemoryRequirements {
    VkDeviceSize    size;
    VkDeviceSize    alignment;
    uint32_t        memoryTypeBits;
} VkMemoryRequirements;
```

Members

- `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.

Description

See Also

`VkDeviceSize`, vkGetBufferMemoryRequirements, vkGetImageMemoryRequirements

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkMemoryRequirements

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkMemoryType(3)

Name

VkMemoryType - Structure specifying memory type

C Specification

The `VkMemoryType` structure is defined as:

```
typedef struct VkMemoryType {
    VkMemoryPropertyFlags    propertyFlags;
    uint32_t                 heapIndex;
} VkMemoryType;
```

Members

- `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.

Description

See Also

VkMemoryPropertyFlags, VkPhysicalDeviceMemoryProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkMemoryType

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkOffset2D(3)

Name

VkOffset2D - Structure specifying a two-dimensional offset

C Specification

A two-dimensional offsets is defined by the structure:

```
typedef struct VkOffset2D {
    int32_t    x;
    int32_t    y;
} VkOffset2D;
```

Members

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

Description

See Also

VkRect2D

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkOffset2D

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkOffset3D(3)

Name

VkOffset3D - Structure specifying a three-dimensional offset

C Specification

A three-dimensional offset is defined by the structure:

```
typedef struct VkOffset3D {
    int32_t    x;
    int32_t    y;
    int32_t    z;
} VkOffset3D;
```

Members

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

Description

See Also

VkBufferImageCopy, VkImageBlit, VkImageCopy, VkImageResolve, VkSparseImageMemoryBind

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkOffset3D

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPhysicalDeviceFeatures(3)

Name

VkPhysicalDeviceFeatures - Structure describing the fine-grained features that can be supported by an implementation

C Specification

The `VkPhysicalDeviceFeatures` structure is defined as:

```
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;
```

Members

The members of the `VkPhysicalDeviceFeatures` structure describe the following features:

Description

- `robustBufferAccess` indicates that accesses to buffers are bounds-checked against the range of the buffer descriptor (as determined by `VkDescriptorBufferInfo`::`range`, `VkBufferViewCreateInfo`::`range`, or the size of the buffer). Out of bounds accesses **must** not cause application termination, and the effects of shader loads, stores, and atomics **must** conform to an implementation-dependent behavior as described below.
  - A buffer access is considered to be out of bounds if any of the following are true:
    - The pointer was formed by `OpImageTexelPointer` and the coordinate is less than zero or greater than or equal to the number of whole elements in the bound range.
    - The pointer was not formed by `OpImageTexelPointer` and the object pointed to is not wholly contained within the bound range.

      > *Note*
      >
      > If a SPIR-V `OpLoad` instruction loads a structure and the tail end of the structure is out of bounds, then all members of the structure are considered out of bounds even if the members at the end are not statically used.

    - If any buffer access in a given SPIR-V block is determined to be out of bounds, then any other access of the same type (load, store, or atomic) in the same SPIR-V block that accesses an address less than 16 bytes away from the out of bounds address **may** also be considered out of bounds.

◦ Out-of-bounds buffer loads will return any of the following values:

  ▪ Values from anywhere within the memory range(s) bound to the buffer (possibly including bytes of memory past the end of the buffer, up to the end of the bound range).

  ▪ Zero values, or (0,0,0,x) vectors for vector reads where x is a valid value represented in the type of the vector components and **may** be any of:

    ▪ 0, 1, or the maximum representable positive integer value, for signed or unsigned integer components

    ▪ 0.0 or 1.0, for floating-point components

◦ Out-of-bounds writes **may** modify values within the memory range(s) bound to the buffer, but **must** not modify any other memory.

◦ Out-of-bounds atomics **may** modify values within the memory range(s) bound to the buffer, but **must** not modify any other memory, and return an undefined value.

◦ Vertex input attributes are considered out of bounds if the address of the attribute plus the size of the attribute is greater than the size of the bound buffer. Further, if any vertex input attribute using a specific vertex input binding is out of bounds, then all vertex input attributes using that vertex input binding for that vertex shader invocation are considered out of bounds.

  ▪ If a vertex input attribute is out of bounds, it will be assigned one of the following values:

    ▪ Values from anywhere within the memory range(s) bound to the buffer, converted according to the format of the attribute.

    ▪ Zero values, format converted according to the format of the attribute.

    ▪ Zero values, or (0,0,0,x) vectors, as described above.

◦ If `robustBufferAccess` is not enabled, out of bounds accesses **may** corrupt any memory within the process and cause undefined behavior up to and including application termination.

- `fullDrawIndexUint32` indicates 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` indicates 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` indicates 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` indicates 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 indicates whether shader modules **can** declare the Geometry capability.

- tessellationShader indicates 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 indicates whether shader modules **can** declare the Tessellation capability.
- sampleRateShading indicates whether per-sample shading and multisample interpolation are supported. If this feature is not enabled, the sampleShadingEnable member of the VkPipelineMultisampleStateCreateInfo structure **must** be set to VK_FALSE and the minSampleShading member is ignored. This also indicates whether shader modules **can** declare the SampleRateShading capability.
- dualSrcBlend indicates 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 html/vkspec.html#framebuffer-dsb.
- logicOp indicates whether logic operations are supported. If this feature is not enabled, the logicOpEnable member of the VkPipelineColorBlendStateCreateInfo structure **must** be set to VK_FALSE, and the logicOp member is ignored.
- multiDrawIndirect indicates 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 indicates whether indirect draw 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 indicates whether depth clamping is supported. If this feature is not enabled, the depthClampEnable member of the VkPipelineRasterizationStateCreateInfo structure **must** be set to VK_FALSE. Otherwise, setting depthClampEnable to VK_TRUE will enable depth clamping.
- depthBiasClamp indicates whether depth bias clamping is supported. If this feature is not enabled, the depthBiasClamp member of the VkPipelineRasterizationStateCreateInfo structure **must** be set to 0.0 unless the VK_DYNAMIC_STATE_DEPTH_BIAS dynamic state is enabled, and the depthBiasClamp parameter to vkCmdSetDepthBias **must** be set to 0.0.
- fillModeNonSolid indicates 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 indicates whether depth bounds tests are supported. If this feature is not enabled, the depthBoundsTestEnable member of the VkPipelineDepthStencilStateCreateInfo structure **must** be set to VK_FALSE. When depthBoundsTestEnable is set to VK_FALSE, the minDepthBounds and maxDepthBounds members of the VkPipelineDepthStencilStateCreateInfo structure are ignored.

- wideLines indicates 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 set to 1.0 unless the VK_DYNAMIC_STATE_LINE_WIDTH dynamic state is enabled, and the lineWidth parameter to vkCmdSetLineWidth **must** be set to 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 indicates 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 indicates whether the implementation is able to replace the alpha value of the color fragment output from the fragment shader with the maximum representable alpha value for fixed-point colors or 1.0 for floating-point colors. If this feature is not enabled, then the alphaToOneEnable member of the VkPipelineMultisampleStateCreateInfo structure **must** be set to VK_FALSE. Otherwise setting alphaToOneEnable to VK_TRUE will enable alpha-to-one behavior.
- multiViewport indicates whether more than one viewport is supported. If this feature is not enabled, the viewportCount and scissorCount members of the VkPipelineViewportStateCreateInfo structure **must** be set to 1. Similarly, the viewportCount parameter to the vkCmdSetViewport command and the scissorCount parameter to the vkCmdSetScissor command **must** be 1, and the firstViewport parameter to the vkCmdSetViewport command and the firstScissor parameter to the vkCmdSetScissor command **must** be 0.
- samplerAnisotropy indicates whether anisotropic filtering is supported. If this feature is not enabled, the maxAnisotropy member of the VkSamplerCreateInfo structure **must** be 1.0.
- textureCompressionETC2 indicates 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

  vkGetPhysicalDeviceFormatProperties and vkGetPhysicalDeviceImageFormatProperties **can** be used to check for additional supported properties of individual formats.

- textureCompressionASTC_LDR indicates 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

vkGetPhysicalDeviceFormatProperties and vkGetPhysicalDeviceImageFormatProperties **can** be used to check for additional supported properties of individual formats.

- textureCompressionBC indicates 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

  vkGetPhysicalDeviceFormatProperties and vkGetPhysicalDeviceImageFormatProperties **can** be used to check for additional supported properties of individual formats.

- occlusionQueryPrecise indicates 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 indicates 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 indicates 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 buffers, and storage buffer variables used by these stages in shader modules **must** be decorated with the NonWriteable decoration (or the readonly memory qualifier in GLSL).
- fragmentStoresAndAtomics indicates 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 buffers, and storage buffer variables used by the fragment stage in shader modules **must** be decorated with the NonWriteable decoration (or the readonly memory qualifier in GLSL).
- shaderTessellationAndGeometryPointSize indicates 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 indicates 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 indicates 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 indicates whether shader modules **can** declare the ImageGatherExtended capability.

- shaderStorageImageExtendedFormats indicates whether the extended storage image formats are available in shader code. If this feature is not enabled, the formats requiring the StorageImageExtendedFormats capability are not supported for storage images. This also indicates whether shader modules **can** declare the StorageImageExtendedFormats capability.
- shaderStorageImageMultisample indicates 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 indicates whether shader modules **can** declare the StorageImageMultisample capability.
- shaderStorageImageReadWithoutFormat indicates whether storage images require a format qualifier to be specified when reading from storage images. If this feature is not enabled, the OpImageRead instruction **must** not have an OpTypeImage of Unknown. This also indicates whether shader modules **can** declare the StorageImageReadWithoutFormat capability.
- shaderStorageImageWriteWithoutFormat indicates whether storage images require a format qualifier to be specified when writing to storage images. If this feature is not enabled, the OpImageWrite instruction **must** not have an OpTypeImage of Unknown. This also indicates whether shader modules **can** declare the StorageImageWriteWithoutFormat capability.
- shaderUniformBufferArrayDynamicIndexing indicates whether arrays of uniform buffers **can** be indexed by *dynamically 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** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the UniformBufferArrayDynamicIndexing capability.
- shaderSampledImageArrayDynamicIndexing indicates whether arrays of samplers or sampled images **can** be indexed by dynamically 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** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the SampledImageArrayDynamicIndexing capability.
- shaderStorageBufferArrayDynamicIndexing indicates whether arrays of storage buffers **can** be indexed by dynamically 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** be indexed only by constant integral expressions when aggregated into arrays in shader code. This also indicates whether shader modules **can** declare the StorageBufferArrayDynamicIndexing capability.
- shaderStorageImageArrayDynamicIndexing indicates whether arrays of storage images **can** be indexed by dynamically uniform integer expressions 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 indicates whether shader modules **can** declare the StorageImageArrayDynamicIndexing capability.
- shaderClipDistance indicates 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 indicates whether shader modules **can** declare the `ClipDistance` capability.

- `shaderCullDistance` indicates 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 indicates whether shader modules **can** declare the `CullDistance` capability.
- `shaderFloat64` indicates 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 indicates whether shader modules **can** declare the `Float64` capability.
- `shaderInt64` indicates 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 indicates whether shader modules **can** declare the `Int64` capability.
- `shaderInt16` indicates 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 indicates whether shader modules **can** declare the `Int16` capability.
- `shaderResourceResidency` indicates 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 indicates whether shader modules **can** declare the `SparseResidency` capability. The feature requires at least one of the `sparseResidency`* features to be supported.
- `shaderResourceMinLod` indicates whether image operations that specify the minimum resource level-of-detail (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 indicates whether shader modules **can** declare the `MinLod` capability.
- `sparseBinding` indicates 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.
- `sparseResidencyBuffer` indicates 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.
- `sparseResidencyImage2D` indicates 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.
- `sparseResidencyImage3D` indicates 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.

- `sparseResidency2Samples` indicates 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.
- `sparseResidency4Samples` indicates 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.
- `sparseResidency8Samples` indicates 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.
- `sparseResidency16Samples` indicates 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.
- `sparseResidencyAliased` indicates 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.
- `variableMultisampleRate` indicates whether all pipelines that will be bound to a command buffer during a subpass with no attachments **must** have the same value for `VkPipelineMultisampleStateCreateInfo`::`rasterizationSamples`. If set to `VK_TRUE`, the implementation supports variable multisample rates in a subpass with 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` indicates whether a secondary command buffer **may** be executed while a query is active.

See Also

`VkBool32`, VkDeviceCreateInfo, vkGetPhysicalDeviceFeatures

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPhysicalDeviceFeatures

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPhysicalDeviceLimits(3)

Name

VkPhysicalDeviceLimits - Structure reporting implementation-dependent physical device limits

C Specification

The VkPhysicalDeviceLimits structure is defined as:

```
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;
```

Members

- `maxImageDimension1D` is the maximum dimension (`width`) supported for all images created with an `imageType` of `VK_IMAGE_TYPE_1D`.
- `maxImageDimension2D` is the maximum dimension (`width` or `height`) supported for all images created with an `imageType` of `VK_IMAGE_TYPE_2D` and without `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT` set in `flags`.
- `maxImageDimension3D` is the maximum dimension (`width`, `height`, or `depth`) supported for all images created with an `imageType` of `VK_IMAGE_TYPE_3D`.
- `maxImageDimensionCube` is the maximum dimension (`width` or `height`) supported for all images created with an `imageType` of `VK_IMAGE_TYPE_2D` and with `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT` set in `flags`.
- `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 any VkDescriptorBufferInfo structures passed to a call 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 any VkDescriptorBufferInfo structures passed to a call 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, (`offset` + `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 size 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 html/vkspec.html#descriptorsets-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. 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 html/vkspec.html#descriptorsets-sampler and html/vkspec.html#descriptorsets-combinedimagesampler.
- 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. 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 html/vkspec.html#descriptorsets-uniformbuffer and html/vkspec.html#descriptorsets-uniformbufferdynamic.
- 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. 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 html/vkspec.html#descriptorsets-storagebuffer and html/vkspec.html#descriptorsets-storagebufferdynamic.
- 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. 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 html/vkspec.html#descriptorsets-combinedimagesampler, html/vkspec.html#descriptorsets-sampledimage, and html/vkspec.html#descriptorsets-uniformtexelbuffer.
- 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. 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 html/vkspec.html#descriptorsets-storageimage, and html/vkspec.html#descriptorsets-storagetexelbuffer.

- 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. 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 html/vkspec.html#descriptorsets-inputattachment.
- 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. 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 descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_SAMPLER or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER count against this limit. See html/vkspec.html#descriptorsets-sampler and html/vkspec.html#descriptorsets-combinedimagesampler.
- maxDescriptorSetUniformBuffers is the maximum number of uniform buffers that **can** be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC count against this limit. See html/vkspec.html#descriptorsets-uniformbuffer and html/vkspec.html#descriptorsets-uniformbufferdynamic.
- maxDescriptorSetUniformBuffersDynamic is the maximum number of dynamic uniform buffers that **can** be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC count against this limit. See html/vkspec.html#descriptorsets-uniformbufferdynamic.
- maxDescriptorSetStorageBuffers is the maximum number of storage buffers that **can** be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC count against this limit. See html/vkspec.html#descriptorsets-storagebuffer and html/vkspec.html#descriptorsets-storagebufferdynamic.
- maxDescriptorSetStorageBuffersDynamic is the maximum number of dynamic storage buffers that **can** be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC count against this limit. See html/vkspec.html#descriptorsets-storagebufferdynamic.
- maxDescriptorSetSampledImages is the maximum number of sampled images that **can** be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. 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. See html/vkspec.html#descriptorsets-combinedimagesampler, html/vkspec.html#descriptorsets-sampledimage, and html/vkspec.html#descriptorsets-uniformtexelbuffer.

- maxDescriptorSetStorageImages is the maximum number of storage images that **can** be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER count against this limit. See html/vkspec.html#descriptorsets-storageimage, and html/vkspec.html#descriptorsets-storagetexelbuffer.
- maxDescriptorSetInputAttachments is the maximum number of input attachments that **can** be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT count against this limit. See html/vkspec.html#descriptorsets-inputattachment.
- 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 html/vkspec.html#fxvertex-attrib and html/vkspec.html#fxvertex-input.
- 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 html/vkspec.html#fxvertex-input.
- 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 html/vkspec.html#fxvertex-input.
- 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 html/vkspec.html#fxvertex-input.
- maxVertexOutputComponents is the maximum number of components of output variables which **can** be output by a vertex shader. See html/vkspec.html#shaders-vertex.
- maxTessellationGenerationLevel is the maximum tessellation generation level supported by the fixed-function tessellation primitive generator. See html/vkspec.html#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 html/vkspec.html#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 html/vkspec.html#geometry.
- 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, 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 html/vkspec.html#framebuffer-dsb and dualSrcBlend.
- maxFragmentCombinedOutputResources is the total number of storage buffers, storage images, and output buffers which **can** be used in the fragment shader stage.
- maxComputeSharedMemorySize is the maximum total storage size, in bytes, of all variables declared with the WorkgroupLocal 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 dispatch 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 dispatch commands **must** be less than or equal to the corresponding limit. See html/vkspec.html#dispatch.
- 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 and 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 specified by the LocalSize execution mode and 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 html/vkspec.html#primsrast.
- subTexelPrecisionBits is the number of bits of precision in the division along an axis of an image used for minification and magnification filters. $2^{\mathtt{subTexelPrecisionBits}}$ is the actual number of divisions along each axis of the image represented. The filtering hardware will snap to these locations when computing 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. $2^{\mathtt{mipmapPrecisionBits}}$ is the actual number of divisions.

> *Note*
>
> For example, if this value is 2 bits then when linearly filtering between two levels, each level could: contribute: 0%, 33%, 66%, or 100% (this is just an example and the amount of contribution **should** be covered by different equations in the spec).

- 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 draw calls. See multiDrawIndirect.
- maxSamplerLodBias is the maximum absolute sampler level of detail 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 [-maxSamplerLodBias,+maxSamplerLodBias]. See html/vkspec.html#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 html/vkspec.html#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 × size, 2 × size - 1], where size =

max(maxViewportDimensions[0], 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 [-size + 1, 2 × 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 html/vkspec.html#memory-device-hostaccess.
- minTexelBufferOffsetAlignment is the minimum **required** alignment, in bytes, for the offset member of the VkBufferViewCreateInfo structure for texel buffers. When a buffer view is created for a buffer which was created with VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT or VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT set in the usage member of the VkBufferCreateInfo structure, the offset **must** be an integer multiple of this limit.
- 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.
- 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.
- 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 or ConstOffsets image operands of any of the OpImage*Gather image instructions.
- maxTexelGatherOffset is the maximum offset value for the Offset or ConstOffsets image operands of any of the OpImage*Gather image instructions.
- minInterpolationOffset is the minimum negative offset value for the offset operand of the InterpolateAtOffset extended instruction.
- maxInterpolationOffset is the maximum positive offset value for the offset operand of the InterpolateAtOffset extended instruction.

- subPixelInterpolationOffsetBits is the number of subpixel 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 bitmask[1] of VkSampleCountFlagBits indicating the color sample counts that are supported for all framebuffer color attachments.
- framebufferDepthSampleCounts is a bitmask[1] of VkSampleCountFlagBits indicating the supported depth sample counts for all framebuffer depth/stencil attachments, when the format includes a depth component.
- framebufferStencilSampleCounts is a bitmask[1] of VkSampleCountFlagBits indicating the supported stencil sample counts for all framebuffer depth/stencil attachments, when the format includes a stencil component.
- framebufferNoAttachmentsSampleCounts is a bitmask[1] of VkSampleCountFlagBits indicating the supported sample counts for a framebuffer with 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 structure **must** be less than or equal to this limit.
- sampledImageColorSampleCounts is a bitmask[1] 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 bitmask[1] 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 bitmask[1] 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 bitmask[1] of VkSampleCountFlagBits indicating the sample 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 bitmask[1] 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 indicates support for timestamps on all graphics and compute queues. If this limit is set to 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 any pair of 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 html/vkspec.html#devsandqueues-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 indicates 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 indicates 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 vkCmdCopyBufferToImage and vkCmdCopyImageToBuffer. The per texel alignment requirements are enforced, but applications **should** use the optimal alignment for optimal performance and power use.
- optimalBufferCopyRowPitchAlignment is the optimal buffer row pitch alignment in bytes for vkCmdCopyBufferToImage 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.
- nonCoherentAtomSize is the size and alignment in bytes that bounds concurrent access to host-

mapped device memory.

Description

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.

See Also

`VkBool32`, `VkDeviceSize`, VkPhysicalDeviceProperties, VkSampleCountFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPhysicalDeviceLimits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPhysicalDeviceMemoryProperties(3)

Name

VkPhysicalDeviceMemoryProperties - Structure specifying physical device memory properties

C Specification

The `VkPhysicalDeviceMemoryProperties` structure is defined as:

```
typedef struct VkPhysicalDeviceMemoryProperties {
    uint32_t        memoryTypeCount;
    VkMemoryType    memoryTypes[VK_MAX_MEMORY_TYPES];
    uint32_t        memoryHeapCount;
    VkMemoryHeap    memoryHeaps[VK_MAX_MEMORY_HEAPS];
} VkPhysicalDeviceMemoryProperties;
```

Members

- `memoryTypeCount` is the number of valid elements in the `memoryTypes` array.
- `memoryTypes` is an array of 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 VkMemoryHeap structures describing the *memory heaps* from which memory **can** be allocated.

Description

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`

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`.

The memory types are sorted according to a preorder which serves to aid in easily selecting an appropriate memory type. Given two memory types X and Y, the preorder defines $X \leq Y$ if:

- the memory property bits set for X are a strict subset of the memory property bits set for Y. Or,
- the memory property bits set for X are the same as the memory property bits set for Y, and X uses a memory heap with greater or equal performance (as determined in an implementation-specific manner).

Memory types are ordered in the list such that X is assigned a lesser `memoryTypeIndex` than Y if $(X \leq Y) \land \neg (Y \leq X)$ according to the preorder. Note that the list of all allowed memory property flag combinations above satisfies this preorder, but other orders would as well. The goal of this ordering is to enable applications to use a simple search loop in selecting the proper memory type, along the lines of:

```
// Find a memory type in "memoryTypeBits" that includes all of "properties"
int32_t FindProperties(uint32_t memoryTypeBits, VkMemoryPropertyFlags properties)
{
    for (int32_t i = 0; i < memoryTypeCount; ++i)
    {
        if ((memoryTypeBits & (1 << i)) &&
            ((memoryTypes[i].propertyFlags & properties) == properties))
            return i;
    }
    return -1;
}

// Try to find an optimal memory type, or if it does not exist
// find any compatible memory type
VkMemoryRequirements memoryRequirements;
vkGetImageMemoryRequirements(device, image, &memoryRequirements);
int32_t memoryType = FindProperties(memoryRequirements.memoryTypeBits,
optimalProperties);
if (memoryType == -1)
    memoryType = FindProperties(memoryRequirements.memoryTypeBits,
requiredProperties);
```

The loop will find the first supported memory type that has all bits requested in `properties` set. If there is no exact match, it will find a closest match (i.e. a memory type with the fewest additional bits set), which has some additional bits set but which are not detrimental to the behaviors requested by `properties`. The application **can** first search for the optimal properties, e.g. a memory type that is device-local or supports coherent cached accesses, as appropriate for the intended usage, and if such a memory type is not present **can** fallback to searching for a less optimal but guaranteed set of properties such as "0" or "host-visible and coherent".

See Also

VkMemoryHeap, VkMemoryType, vkGetPhysicalDeviceMemoryProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPhysicalDeviceMemoryProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPhysicalDeviceProperties(3)

Name

VkPhysicalDeviceProperties - Structure specifying physical device properties

C Specification

The `VkPhysicalDeviceProperties` structure is defined as:

```
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;
```

Members

- `apiVersion` is the version of Vulkan supported by the device, encoded as described in the API Version Numbers and Semantics section.
- `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 a null-terminated UTF-8 string containing the name of the device.
- `pipelineCacheUUID` is an array of size `VK_UUID_SIZE`, containing 8-bit values that represent a universally unique identifier for the device.
- `limits` is the VkPhysicalDeviceLimits structure which specifies device-specific limits of the physical device. See Limits for details.
- `sparseProperties` is the VkPhysicalDeviceSparseProperties structure which specifies various sparse related properties of the physical device. See Sparse Properties for details.

Description

The `vendorID` and `deviceID` fields are provided to allow applications to adapt to device characteristics that are not adequately exposed by other Vulkan queries. These **may** include performance profiles, hardware errata, or other characteristics. In PCI-based implementations, the low sixteen bits of `vendorID` and `deviceID` **must** contain (respectively) the PCI vendor and device IDs associated with the hardware device, and the remaining bits **must** be set to zero. In non-PCI implementations, the choice of what values to return **may** be dictated by operating system or platform policies. It is otherwise at the discretion of the implementer, subject to the following constraints and guidelines:

- For purposes of physical device identification, the *vendor* of a physical device is the entity responsible for the most salient characteristics of the hardware represented by the physical device handle. In the case of a discrete GPU, this **should** be the GPU chipset vendor. In the case of a GPU or other accelerator integrated into a system-on-chip (SoC), this **should** be the supplier of the silicon IP used to create the GPU or other accelerator.
- If the vendor of the physical device has a valid PCI vendor ID issued by PCI-SIG, that ID **should** be used to construct `vendorID` as described above for PCI-based implementations. Implementations that do not return a PCI vendor ID in `vendorID` **must** return a valid Khronos vendor ID, obtained as described in the Vulkan Documentation and Extensions 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.
- The vendor of the physical device is responsible for selecting `deviceID`. The value selected **should** uniquely identify both the device version and any major configuration options (for example, core count in the case of multicore devices). 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.

See Also

VkPhysicalDeviceLimits, VkPhysicalDeviceSparseProperties, VkPhysicalDeviceType, vkGetPhysicalDeviceProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPhysicalDeviceProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPhysicalDeviceSparseProperties(3)

Name

VkPhysicalDeviceSparseProperties - Structure specifying physical device sparse memory properties

C Specification

The VkPhysicalDeviceSparseProperties structure is defined as:

```
typedef struct VkPhysicalDeviceSparseProperties {
    VkBool32    residencyStandard2DBlockShape;
    VkBool32    residencyStandard2DMultisampleBlockShape;
    VkBool32    residencyStandard3DBlockShape;
    VkBool32    residencyAlignedMipSize;
    VkBool32    residencyNonResidentStrict;
} VkPhysicalDeviceSparseProperties;
```

Members

- residencyStandard2DBlockShape is VK_TRUE if the physical device will access all single-sample 2D sparse resources using the standard sparse image block shapes (based on image format), as described in the Standard Sparse Image Block Shapes (Single Sample) table. If this property is not supported the value returned in the imageGranularity member of the VkSparseImageFormatProperties structure for single-sample 2D images is not **required** to match the standard sparse image block dimensions listed in the table.
- residencyStandard2DMultisampleBlockShape is VK_TRUE if the physical device will access all multisample 2D sparse resources using the standard sparse image block shapes (based on image format), as described in the Standard Sparse Image Block Shapes (MSAA) table. If this property is not supported, the value returned in the imageGranularity member of the VkSparseImageFormatProperties structure for multisample 2D images is not **required** to match the standard sparse image block dimensions listed in the table.
- residencyStandard3DBlockShape is VK_TRUE if the physical device will access all 3D sparse resources using the standard sparse image block shapes (based on image format), as described in the Standard Sparse Image Block Shapes (Single Sample) table. If this property is not supported, the value returned in the imageGranularity member of the VkSparseImageFormatProperties structure for 3D images is not **required** to match the standard sparse image block dimensions listed in the table.
- residencyAlignedMipSize is VK_TRUE if images with mip level dimensions that are not integer multiples of the corresponding dimensions of the sparse image block **may** be placed in the mip tail. If this property is not reported, only mip levels with dimensions smaller than the imageGranularity member of the VkSparseImageFormatProperties structure will be placed in the mip tail. If this property is reported the implementation is allowed to return VK_SPARSE_IMAGE_FORMAT_ALIGNED_MIP_SIZE_BIT in the flags member of VkSparseImageFormatProperties, indicating that mip level dimensions that are not integer multiples of the corresponding dimensions of the sparse image block will be placed in the mip

tail.

* `residencyNonResidentStrict` specifies whether the physical device **can** consistently access non-resident regions of a resource. If this property is `VK_TRUE`, access to non-resident regions of resources will be guaranteed to return values as if the resource were populated with 0; writes to non-resident regions will be discarded.

Description

See Also

`VkBool32`, VkPhysicalDeviceProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPhysicalDeviceSparseProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineCacheCreateInfo(3)

Name

VkPipelineCacheCreateInfo - Structure specifying parameters of a newly created pipeline cache

C Specification

The `VkPipelineCacheCreateInfo` structure is defined as:

```
typedef struct VkPipelineCacheCreateInfo {
    VkStructureType               sType;
    const void*                   pNext;
    VkPipelineCacheCreateFlags    flags;
    size_t                        initialDataSize;
    const void*                   pInitialData;
} VkPipelineCacheCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `initialDataSize` is the number of bytes in `pInitialData`. If `initialDataSize` is zero, the pipeline cache will initially be empty.
- `pInitialData` is a pointer to previously retrieved pipeline cache data. If the pipeline cache data is incompatible (as defined below) with the device, the pipeline cache will be initially empty. If `initialDataSize` is zero, `pInitialData` is ignored.

Description

Valid Usage

- If `initialDataSize` is not `0`, it **must** be equal to the size of `pInitialData`, as returned by `vkGetPipelineCacheData` when `pInitialData` was originally retrieved
- If `initialDataSize` is not `0`, `pInitialData` **must** have been retrieved from a previous call to `vkGetPipelineCacheData`

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_CACHE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- If `initialDataSize` is not `0`, `pInitialData` **must** be a pointer to an array of `initialDataSize` bytes

See Also

VkPipelineCacheCreateFlags, VkStructureType, vkCreatePipelineCache

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineCacheCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineColorBlendAttachmentState(3)

Name

VkPipelineColorBlendAttachmentState - Structure specifying a pipeline color blend attachment state

C Specification

The `VkPipelineColorBlendAttachmentState` structure is defined as:

```
typedef struct VkPipelineColorBlendAttachmentState {
    VkBool32                 blendEnable;
    VkBlendFactor            srcColorBlendFactor;
    VkBlendFactor            dstColorBlendFactor;
    VkBlendOp                colorBlendOp;
    VkBlendFactor            srcAlphaBlendFactor;
    VkBlendFactor            dstAlphaBlendFactor;
    VkBlendOp                alphaBlendOp;
    VkColorComponentFlags    colorWriteMask;
} VkPipelineColorBlendAttachmentState;
```

Members

- `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 use 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.

Description

Valid Usage

- If the dual source blending 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
- If the dual source blending 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
- If the dual source blending 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
- If the dual source blending 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

Valid Usage (Implicit)

- srcColorBlendFactor **must** be a valid VkBlendFactor value
- dstColorBlendFactor **must** be a valid VkBlendFactor value
- colorBlendOp **must** be a valid VkBlendOp value
- srcAlphaBlendFactor **must** be a valid VkBlendFactor value
- dstAlphaBlendFactor **must** be a valid VkBlendFactor value
- alphaBlendOp **must** be a valid VkBlendOp value
- colorWriteMask **must** be a valid combination of VkColorComponentFlagBits values

See Also

VkBlendFactor, VkBlendOp, VkBool32, VkColorComponentFlags, VkPipelineColorBlendStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineColorBlendAttachmentState

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineColorBlendStateCreateInfo(3)

Name

VkPipelineColorBlendStateCreateInfo - Structure specifying parameters of a newly created pipeline color blend state

C Specification

The `VkPipelineColorBlendStateCreateInfo` structure is defined as:

```
typedef struct VkPipelineColorBlendStateCreateInfo {
    VkStructureType                               sType;
    const void*                                   pNext;
    VkPipelineColorBlendStateCreateFlags          flags;
    VkBool32                                      logicOpEnable;
    VkLogicOp                                     logicOp;
    uint32_t                                      attachmentCount;
    const VkPipelineColorBlendAttachmentState*    pAttachments;
    float                                         blendConstants[4];
} VkPipelineColorBlendStateCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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`. This value **must** equal the `colorAttachmentCount` for the subpass in which this pipeline is used.
- `pAttachments`: is a pointer to array of per target attachment states.
- `blendConstants` is 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.

Description

Each element of the `pAttachments` array is a VkPipelineColorBlendAttachmentState structure specifying per-target blending state for each individual color attachment. If the independent blending feature is not enabled on the device, all VkPipelineColorBlendAttachmentState elements in the `pAttachments` array **must** be identical.

Valid Usage

- If the independent blending feature is not enabled, all elements of pAttachments **must** be identical
- If the logic operations feature is not enabled, logicOpEnable **must** be VK_FALSE
- If logicOpEnable is VK_TRUE, logicOp **must** be a valid VkLogicOp value

Valid Usage (Implicit)

- sType **must** be VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO
- pNext **must** be NULL
- flags **must** be 0
- If attachmentCount is not 0, pAttachments **must** be a pointer to an array of attachmentCount valid VkPipelineColorBlendAttachmentState structures

See Also

VkBool32, VkGraphicsPipelineCreateInfo, VkLogicOp, VkPipelineColorBlendAttachmentState, VkPipelineColorBlendStateCreateFlags, VkStructureType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineColorBlendStateCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineDepthStencilStateCreateInfo(3)

Name

VkPipelineDepthStencilStateCreateInfo - Structure specifying parameters of a newly created pipeline depth stencil state

C Specification

The `VkPipelineDepthStencilStateCreateInfo` structure is defined as:

```
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;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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 the comparison operator used in the depth test.
- `depthBoundsTestEnable` controls whether depth bounds testing is enabled.
- `stencilTestEnable` controls whether stencil testing is enabled.
- `front` and `back` control the parameters of the stencil test.
- `minDepthBounds` and `maxDepthBounds` define the range of values used in the depth bounds test.

Description

Valid Usage

- If the depth bounds testing feature is not enabled, `depthBoundsTestEnable` **must** be `VK_FALSE`

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `depthCompareOp` **must** be a valid VkCompareOp value
- `front` **must** be a valid `VkStencilOpState` structure
- `back` **must** be a valid `VkStencilOpState` structure

See Also

`VkBool32`, VkCompareOp, VkGraphicsPipelineCreateInfo, VkPipelineDepthStencilStateCreateFlags, VkStencilOpState, VkStructureType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineDepthStencilStateCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineDynamicStateCreateInfo(3)

Name

VkPipelineDynamicStateCreateInfo - Structure specifying parameters of a newly created pipeline dynamic state

C Specification

The `VkPipelineDynamicStateCreateInfo` structure is defined as:

```
typedef struct VkPipelineDynamicStateCreateInfo {
    VkStructureType                      sType;
    const void*                          pNext;
    VkPipelineDynamicStateCreateFlags    flags;
    uint32_t                             dynamicStateCount;
    const VkDynamicState*                pDynamicStates;
} VkPipelineDynamicStateCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `dynamicStateCount` is the number of elements in the `pDynamicStates` array.
- `pDynamicStates` is 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 info.

Description

Valid Usage

- Each element of `pDynamicStates` **must** be unique

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `pDynamicStates` **must** be a pointer to an array of `dynamicStateCount` valid VkDynamicState values
- `dynamicStateCount` **must** be greater than `0`

See Also

VkDynamicState, VkGraphicsPipelineCreateInfo, VkPipelineDynamicStateCreateFlags, VkStructureType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineDynamicStateCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineInputAssemblyStateCreateInfo(3)

Name

VkPipelineInputAssemblyStateCreateInfo - Structure specifying parameters of a newly created pipeline input assembly state

C Specification

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`:

```
typedef struct VkPipelineInputAssemblyStateCreateInfo {
    VkStructureType                            sType;
    const void*                                pNext;
    VkPipelineInputAssemblyStateCreateFlags    flags;
    VkPrimitiveTopology                        topology;
    VkBool32                                   primitiveRestartEnable;
} VkPipelineInputAssemblyStateCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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`, or 0xFFFF when `indexType` is equal to `VK_INDEX_TYPE_UINT16`. Primitive restart is not allowed for “list” topologies.

Description

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

- 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, VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY or VK_PRIMITIVE_TOPOLOGY_PATCH_LIST, primitiveRestartEnable **must** be VK_FALSE
- If the geometry shaders 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
- If the tessellation shaders feature is not enabled, topology **must** not be VK_PRIMITIVE_TOPOLOGY_PATCH_LIST

Valid Usage (Implicit)

- sType **must** be VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO
- pNext **must** be NULL
- flags **must** be 0
- topology **must** be a valid VkPrimitiveTopology value

See Also

VkBool32, VkGraphicsPipelineCreateInfo, VkPipelineInputAssemblyStateCreateFlags, VkPrimitiveTopology, VkStructureType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineInputAssemblyStateCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineLayoutCreateInfo(3)

Name

VkPipelineLayoutCreateInfo - Structure specifying the parameters of a newly created pipeline layout object

C Specification

The VkPipelineLayoutCreateInfo structure is defined as:

```
typedef struct VkPipelineLayoutCreateInfo {
    VkStructureType                 sType;
    const void*                     pNext;
    VkPipelineLayoutCreateFlags     flags;
    uint32_t                        setLayoutCount;
    const VkDescriptorSetLayout*    pSetLayouts;
    uint32_t                        pushConstantRangeCount;
    const VkPushConstantRange*      pPushConstantRanges;
} VkPipelineLayoutCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `setLayoutCount` is the number of descriptor sets included in the pipeline layout.
- `pSetLayouts` is a pointer to an array of `VkDescriptorSetLayout` objects.
- `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.

Description

Valid Usage

- setLayoutCount **must** be less than or equal to VkPhysicalDeviceLimits::maxBoundDescriptorSets
- The total number of descriptors of the type 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
- The total number of descriptors of the type 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
- The total number of descriptors of the type 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
- 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 to any given shader stage across all elements of pSetLayouts **must** be less than or equal to VkPhysicalDeviceLimits::maxPerStageDescriptorSampledImages
- The total number of descriptors of the type 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
- Any two elements of pPushConstantRanges **must** not include the same stage in stageFlags

Valid Usage (Implicit)

- sType **must** be VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO
- pNext **must** be NULL
- flags **must** be 0
- If setLayoutCount is not 0, pSetLayouts **must** be a pointer to an array of setLayoutCount valid VkDescriptorSetLayout handles
- If pushConstantRangeCount is not 0, pPushConstantRanges **must** be a pointer to an array of pushConstantRangeCount valid VkPushConstantRange structures

See Also

VkDescriptorSetLayout, VkPipelineLayoutCreateFlags, VkPushConstantRange, VkStructureType, vkCreatePipelineLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineLayoutCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineMultisampleStateCreateInfo(3)

Name

VkPipelineMultisampleStateCreateInfo - Structure specifying parameters of a newly created pipeline multisample state

C Specification

The VkPipelineMultisampleStateCreateInfo structure is defined as:

```
typedef struct VkPipelineMultisampleStateCreateInfo {
    VkStructureType                          sType;
    const void*                              pNext;
    VkPipelineMultisampleStateCreateFlags    flags;
    VkSampleCountFlagBits                    rasterizationSamples;
    VkBool32                                 sampleShadingEnable;
    float                                    minSampleShading;
    const VkSampleMask*                      pSampleMask;
    VkBool32                                 alphaToCoverageEnable;
    VkBool32                                 alphaToOneEnable;
} VkPipelineMultisampleStateCreateInfo;
```

Members

- sType is the type of this structure.
- pNext is NULL or a pointer to an extension-specific structure.
- flags is reserved for future use.
- rasterizationSamples is a VkSampleCountFlagBits specifying the number of samples per pixel used in rasterization.
- sampleShadingEnable specifies that fragment shading executes per-sample if VK_TRUE, or per-fragment if VK_FALSE, as described in Sample Shading.
- minSampleShading is the minimum fraction of sample shading, as described in Sample Shading.
- pSampleMask is a bitmask of static coverage information that is ANDed with the coverage information generated during rasterization, as described in Sample Mask.
- 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.

Description

Valid Usage

- If the sample rate shading feature is not enabled, `sampleShadingEnable` **must** be `VK_FALSE`
- If the alpha to one feature is not enabled, `alphaToOneEnable` **must** be `VK_FALSE`
- `minSampleShading` **must** be in the range [0,1]

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `rasterizationSamples` **must** be a valid VkSampleCountFlagBits value
- If `pSampleMask` is not `NULL`, `pSampleMask` **must** be a pointer to an array of $\lceil \frac{rasterizationSamples}{32} \rceil$ `VkSampleMask` values

See Also

`VkBool32`, VkGraphicsPipelineCreateInfo, VkPipelineMultisampleStateCreateFlags, VkSampleCountFlagBits, `VkSampleMask`, VkStructureType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineMultisampleStateCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineRasterizationStateCreateInfo(3)

Name

VkPipelineRasterizationStateCreateInfo - Structure specifying parameters of a newly created pipeline rasterization state

C Specification

The `VkPipelineRasterizationStateCreateInfo` structure is defined as:

```
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;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `depthClampEnable` controls whether to clamp the fragment's depth values instead of clipping primitives to the z planes of the frustum, as described in Primitive Clipping.
- `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.

Description

Valid Usage

- If the depth clamping feature is not enabled, `depthClampEnable` **must** be `VK_FALSE`
- If the non-solid fill modes feature is not enabled, `polygonMode` **must** be `VK_POLYGON_MODE_FILL`

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `polygonMode` **must** be a valid VkPolygonMode value
- `cullMode` **must** be a valid combination of VkCullModeFlagBits values
- `frontFace` **must** be a valid VkFrontFace value

See Also

`VkBool32`, VkCullModeFlags, VkFrontFace, VkGraphicsPipelineCreateInfo, VkPipelineRasterizationStateCreateFlags, VkPolygonMode, VkStructureType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineRasterizationStateCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineShaderStageCreateInfo(3)

Name

VkPipelineShaderStageCreateInfo - Structure specifying parameters of a newly created pipeline shader stage

C Specification

The `VkPipelineShaderStageCreateInfo` structure is defined as:

```
typedef struct VkPipelineShaderStageCreateInfo {
    VkStructureType                     sType;
    const void*                         pNext;
    VkPipelineShaderStageCreateFlags    flags;
    VkShaderStageFlagBits               stage;
    VkShaderModule                      module;
    const char*                         pName;
    const VkSpecializationInfo*         pSpecializationInfo;
} VkPipelineShaderStageCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `stage` is a VkShaderStageFlagBits value specifying a single pipeline stage.
- `module` is a `VkShaderModule` object that contains the shader for this stage.
- `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 VkSpecializationInfo, as described in Specialization Constants, and **can** be `NULL`.

Description

Valid Usage

- If the geometry shaders feature is not enabled, stage **must** not be VK_SHADER_STAGE_GEOMETRY_BIT
- If the tessellation shaders feature is not enabled, stage **must** not be VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT or VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
- stage **must** not be VK_SHADER_STAGE_ALL_GRAPHICS, or VK_SHADER_STAGE_ALL
- pName **must** be the name of an OpEntryPoint in module with an execution model that matches stage
- 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
- 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
- If the identified entry point includes any variables in its interface that are declared with the ClipDistance or CullDistance BuiltIn decoration, those variables **must** not have array sizes which sum to more than VkPhysicalDeviceLimits::maxCombinedClipAndCullDistances
- 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
- If stage is VK_SHADER_STAGE_VERTEX_BIT, the identified entry point **must** not include any input variable in its interface that is decorated with CullDistance
- 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 that specifies a patch size with OutputVertices, the patch size **must** be greater than 0 and less than or equal to VkPhysicalDeviceLimits::maxTessellationPatchSize
- If stage is VK_SHADER_STAGE_GEOMETRY_BIT, the identified entry point **must** have an OpExecutionMode instruction that specifies a maximum output vertex count that is greater than 0 and less than or equal to VkPhysicalDeviceLimits::maxGeometryOutputVertices
- If stage is VK_SHADER_STAGE_GEOMETRY_BIT, the identified entry point **must** have an OpExecutionMode instruction that specifies an invocation count that is greater than 0 and less than or equal to VkPhysicalDeviceLimits::maxGeometryShaderInvocations
- If stage is 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
- If stage is 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
- If stage is VK_SHADER_STAGE_FRAGMENT_BIT, the identified entry point **must** not include any output variables in its interface decorated with CullDistance

- If `stage` is `VK_SHADER_STAGE_FRAGMENT_BIT`, and the identified entry point writes to `FragDepth` in any execution path, it **must** write to `FragDepth` in all execution paths

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `stage` **must** be a valid VkShaderStageFlagBits value
- `module` **must** be a valid `VkShaderModule` handle
- `pName` **must** be a null-terminated UTF-8 string
- If `pSpecializationInfo` is not `NULL`, `pSpecializationInfo` **must** be a pointer to a valid `VkSpecializationInfo` structure

See Also

VkComputePipelineCreateInfo, VkGraphicsPipelineCreateInfo, VkPipelineShaderStageCreateFlags, VkShaderModule, VkShaderStageFlagBits, VkSpecializationInfo, VkStructureType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineShaderStageCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineTessellationStateCreateInfo(3)

Name

VkPipelineTessellationStateCreateInfo - Structure specifying parameters of a newly created pipeline tessellation state

C Specification

The `VkPipelineTessellationStateCreateInfo` structure is defined as:

```
typedef struct VkPipelineTessellationStateCreateInfo {
    VkStructureType                           sType;
    const void*                               pNext;
    VkPipelineTessellationStateCreateFlags    flags;
    uint32_t                                  patchControlPoints;
} VkPipelineTessellationStateCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `patchControlPoints` number of control points per patch.

Description

Valid Usage

- `patchControlPoints` **must** be greater than zero and less than or equal to `VkPhysicalDeviceLimits`::`maxTessellationPatchSize`

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_STATE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`

See Also

VkGraphicsPipelineCreateInfo, VkPipelineTessellationStateCreateFlags, VkStructureType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineTessellationStateCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineVertexInputStateCreateInfo(3)

Name

VkPipelineVertexInputStateCreateInfo - Structure specifying parameters of a newly created pipeline vertex input state

C Specification

The `VkPipelineVertexInputStateCreateInfo` structure is defined as:

```
typedef struct VkPipelineVertexInputStateCreateInfo {
    VkStructureType                             sType;
    const void*                                 pNext;
    VkPipelineVertexInputStateCreateFlags       flags;
    uint32_t                                    vertexBindingDescriptionCount;
    const VkVertexInputBindingDescription*      pVertexBindingDescriptions;
    uint32_t                                    vertexAttributeDescriptionCount;
    const VkVertexInputAttributeDescription*    pVertexAttributeDescriptions;
} VkPipelineVertexInputStateCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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.

Description

Valid Usage

- `vertexBindingDescriptionCount` **must** be less than or equal to `VkPhysicalDeviceLimits` ::`maxVertexInputBindings`
- `vertexAttributeDescriptionCount` **must** be less than or equal to `VkPhysicalDeviceLimits` ::`maxVertexInputAttributes`
- For every `binding` specified by any given element of `pVertexAttributeDescriptions`, a `VkVertexInputBindingDescription` **must** exist in `pVertexBindingDescriptions` with the same value of `binding`
- All elements of `pVertexBindingDescriptions` **must** describe distinct binding numbers
- All elements of `pVertexAttributeDescriptions` **must** describe distinct attribute locations

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- If `vertexBindingDescriptionCount` is not `0`, `pVertexBindingDescriptions` **must** be a pointer to an array of `vertexBindingDescriptionCount` valid `VkVertexInputBindingDescription` structures
- If `vertexAttributeDescriptionCount` is not `0`, `pVertexAttributeDescriptions` **must** be a pointer to an array of `vertexAttributeDescriptionCount` valid `VkVertexInputAttributeDescription` structures

See Also

VkGraphicsPipelineCreateInfo, VkPipelineVertexInputStateCreateFlags, VkStructureType, VkVertexInputAttributeDescription, VkVertexInputBindingDescription

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineVertexInputStateCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineViewportStateCreateInfo(3)

Name

VkPipelineViewportStateCreateInfo - Structure specifying parameters of a newly created pipeline viewport state

C Specification

The `VkPipelineViewportStateCreateInfo` structure is defined as:

```
typedef struct VkPipelineViewportStateCreateInfo {
    VkStructureType                       sType;
    const void*                           pNext;
    VkPipelineViewportStateCreateFlags    flags;
    uint32_t                              viewportCount;
    const VkViewport*                     pViewports;
    uint32_t                              scissorCount;
    const VkRect2D*                       pScissors;
} VkPipelineViewportStateCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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 which define the rectangular bounds of the scissor for the corresponding viewport. If the scissor state is dynamic, this member is ignored.

Description

Valid Usage

- If the multiple viewports feature is not enabled, `viewportCount` **must** be `1`
- If the multiple viewports feature is not enabled, `scissorCount` **must** be `1`
- `viewportCount` **must** be between `1` and `VkPhysicalDeviceLimits`::`maxViewports`, inclusive
- `scissorCount` **must** be between `1` and `VkPhysicalDeviceLimits`::`maxViewports`, inclusive
- `scissorCount` and `viewportCount` **must** be identical

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `viewportCount` **must** be greater than `0`
- `scissorCount` **must** be greater than `0`

See Also

VkGraphicsPipelineCreateInfo, VkPipelineViewportStateCreateFlags, VkRect2D, VkStructureType, VkViewport

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineViewportStateCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPushConstantRange(3)

Name

VkPushConstantRange - Structure specifying a push constant range

C Specification

The `VkPushConstantRange` structure is defined as:

```
typedef struct VkPushConstantRange {
    VkShaderStageFlags    stageFlags;
    uint32_t              offset;
    uint32_t              size;
} VkPushConstantRange;
```

Members

- `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 result in undefined data being read.
- `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.

Description

Valid Usage

- `offset` **must** be less than `VkPhysicalDeviceLimits`::`maxPushConstantsSize`
- `offset` **must** be a multiple of `4`
- `size` **must** be greater than `0`
- `size` **must** be a multiple of `4`
- `size` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxPushConstantsSize` minus `offset`

Valid Usage (Implicit)

- `stageFlags` **must** be a valid combination of VkShaderStageFlagBits values
- `stageFlags` **must** not be `0`

See Also

VkPipelineLayoutCreateInfo, VkShaderStageFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPushConstantRange

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkQueryPoolCreateInfo(3)

Name

VkQueryPoolCreateInfo - Structure specifying parameters of a newly created query pool

C Specification

The VkQueryPoolCreateInfo structure is defined as:

```
typedef struct VkQueryPoolCreateInfo {
    VkStructureType                  sType;
    const void*                      pNext;
    VkQueryPoolCreateFlags           flags;
    VkQueryType                      queryType;
    uint32_t                         queryCount;
    VkQueryPipelineStatisticFlags    pipelineStatistics;
} VkQueryPoolCreateInfo;
```

Members

- sType is the type of this structure.
- pNext is NULL or a pointer to an extension-specific structure.
- flags is reserved for future use.
- 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 html/vkspec.html#queries-pipestats.

Description

pipelineStatistics is ignored if queryType is not VK_QUERY_TYPE_PIPELINE_STATISTICS.

Valid Usage

- If the pipeline statistics queries feature is not enabled, queryType **must** not be VK_QUERY_TYPE_PIPELINE_STATISTICS
- If queryType is VK_QUERY_TYPE_PIPELINE_STATISTICS, pipelineStatistics **must** be a valid combination of VkQueryPipelineStatisticFlagBits values

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `queryType` **must** be a valid VkQueryType value

See Also

VkQueryPipelineStatisticFlags, VkQueryPoolCreateFlags, VkQueryType, VkStructureType, vkCreateQueryPool

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueryPoolCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkQueueFamilyProperties(3)

Name

VkQueueFamilyProperties - Structure providing information about a queue family

C Specification

The `VkQueueFamilyProperties` structure is defined as:

```
typedef struct VkQueueFamilyProperties {
    VkQueueFlags    queueFlags;
    uint32_t        queueCount;
    uint32_t        timestampValidBits;
    VkExtent3D      minImageTransferGranularity;
} VkQueueFamilyProperties;
```

Members

- `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.
- `timestampValidBits` is the unsigned integer count of meaningful bits in the timestamps written via `vkCmdWriteTimestamp`. The valid range for the count is 36..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.

Description

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) which indicates 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.

See Also

VkExtent3D, VkQueueFlags, vkGetPhysicalDeviceQueueFamilyProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueueFamilyProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkRect2D(3)

Name

VkRect2D - Structure specifying a two-dimensional subregion

C Specification

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

```
typedef struct VkRect2D {
    VkOffset2D    offset;
    VkExtent2D    extent;
} VkRect2D;
```

Members

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

Description

See Also

VkClearRect, VkExtent2D, VkOffset2D, VkPipelineViewportStateCreateInfo, VkRenderPassBeginInfo, vkCmdSetScissor

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkRect2D

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkRenderPassBeginInfo(3)

Name

VkRenderPassBeginInfo - Structure specifying render pass begin info

C Specification

The `VkRenderPassBeginInfo` structure is defined as:

```
typedef struct VkRenderPassBeginInfo {
    VkStructureType        sType;
    const void*            pNext;
    VkRenderPass           renderPass;
    VkFramebuffer          framebuffer;
    VkRect2D               renderArea;
    uint32_t               clearValueCount;
    const VkClearValue*    pClearValues;
} VkRenderPassBeginInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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 an array of VkClearValue structures that contains 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.

Description

`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, otherwise the pixels outside of the render area become undefined and shader side effects **may** occur for fragments outside 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

- `clearValueCount` **must** be greater than the largest attachment index in `renderPass` that specifies a `loadOp` (or `stencilLoadOp`, if the attachment has a depth/stencil format) of `VK_ATTACHMENT_LOAD_OP_CLEAR`
- If `clearValueCount` is not `0`, `pClearValues` **must** be a pointer to an array of `clearValueCount` valid `VkClearValue` unions
- `renderPass` **must** be compatible with the `renderPass` member of the `VkFramebufferCreateInfo` structure specified when creating `framebuffer`.

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO`
- `pNext` **must** be `NULL`
- `renderPass` **must** be a valid `VkRenderPass` handle
- `framebuffer` **must** be a valid `VkFramebuffer` handle
- Both of `framebuffer`, and `renderPass` **must** have been created, allocated, or retrieved from the same `VkDevice`

See Also

VkClearValue, VkFramebuffer, VkRect2D, VkRenderPass, VkStructureType, vkCmdBeginRenderPass

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkRenderPassBeginInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkRenderPassCreateInfo(3)

Name

VkRenderPassCreateInfo - Structure specifying parameters of a newly created render pass

C Specification

The `VkRenderPassCreateInfo` structure is defined as:

```
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;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `attachmentCount` is the number of attachments used by this render pass, or zero indicating no attachments. Attachments are referred to by zero-based indices in the range [0,`attachmentCount`).
- `pAttachments` points to an array of `attachmentCount` number of VkAttachmentDescription structures describing properties of the attachments, or `NULL` if `attachmentCount` is zero.
- `subpassCount` is the number of subpasses to create for this render pass. Subpasses are referred to by zero-based indices in the range [0,`subpassCount`). A render pass **must** have at least one subpass.
- `pSubpasses` points to an array of `subpassCount` number of VkSubpassDescription structures describing properties of the subpasses.
- `dependencyCount` is the number of dependencies between pairs of subpasses, or zero indicating no dependencies.
- `pDependencies` points to an array of `dependencyCount` number of VkSubpassDependency structures describing dependencies between pairs of subpasses, or `NULL` if `dependencyCount` is zero.

Description

Valid Usage

- 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
- If the attachment member of any element of pInputAttachments, pColorAttachments, pResolveAttachments or pDepthStencilAttachment, or the attachment indexed by any element of pPreserveAttachments in any given 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 VkAttachmentDescription structures describing them **must** include VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT in flags
- If the attachment member of any element of pInputAttachments, pColorAttachments, pResolveAttachments or pDepthStencilAttachment, or any element of pPreserveAttachments in any given element of pSubpasses is not VK_ATTACHMENT_UNUSED, it **must** be less than attachmentCount
- The value of any element of the pPreserveAttachments member in any given element of pSubpasses **must** not be VK_ATTACHMENT_UNUSED
- 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.
- 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 a pipeline stage supported by the pipeline identified by the pipelineBindPoint member of the source subpass.
- 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 a pipeline stage supported by the pipeline identified by the pipelineBindPoint member of the source subpass.

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- If `attachmentCount` is not `0`, `pAttachments` **must** be a pointer to an array of `attachmentCount` valid `VkAttachmentDescription` structures
- `pSubpasses` **must** be a pointer to an array of `subpassCount` valid `VkSubpassDescription` structures
- If `dependencyCount` is not `0`, `pDependencies` **must** be a pointer to an array of `dependencyCount` valid `VkSubpassDependency` structures
- `subpassCount` **must** be greater than `0`

See Also

VkAttachmentDescription, VkRenderPassCreateFlags, VkStructureType, VkSubpassDependency, VkSubpassDescription, vkCreateRenderPass

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkRenderPassCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSamplerCreateInfo(3)

Name

VkSamplerCreateInfo - Structure specifying parameters of a newly created sampler

C Specification

The `VkSamplerCreateInfo` structure is defined as:

```
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;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `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 outside [0..1] range for U coordinate.
- `addressModeV` is a VkSamplerAddressMode value specifying the addressing mode for outside [0..1] range for V coordinate.
- `addressModeW` is a VkSamplerAddressMode value specifying the addressing mode for outside [0..1] range for W coordinate.

- 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 Level-of-Detail 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.
- 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 function to apply to fetched data before filtering as described in the Depth Compare Operation section.
- minLod and maxLod are the values used to clamp the computed level-of-detail value, as described in the Level-of-Detail Operation section. maxLod **must** be greater than or equal to minLod.
- 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 set to VK_TRUE, the range of the image coordinates used to lookup the texel is in the range of zero to the image dimensions for x, y and z. When set to VK_FALSE the range of image coordinates is zero to one. When unnormalizedCoordinates is VK_TRUE, samplers have the following requirements:
  - minFilter and magFilter **must** be equal.
  - mipmapMode **must** be VK_SAMPLER_MIPMAP_MODE_NEAREST.
  - minLod and maxLod **must** be zero.
  - addressModeU and addressModeV **must** each be either VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE or VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER.
  - anisotropyEnable **must** be VK_FALSE.
  - compareEnable **must** be VK_FALSE.
- 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.

Description

> *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. If `maxSamplerAllocationCount` is exceeded, `vkCreateSampler` will return `VK_ERROR_TOO_MANY_OBJECTS`.

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

- The absolute value of mipLodBias **must** be less than or equal to VkPhysicalDeviceLimits ::maxSamplerLodBias
- If the anisotropic sampling feature is not enabled, anisotropyEnable **must** be VK_FALSE
- If anisotropyEnable is VK_TRUE, maxAnisotropy **must** be between 1.0 and VkPhysicalDeviceLimits::maxSamplerAnisotropy, inclusive
- If unnormalizedCoordinates is VK_TRUE, minFilter and magFilter **must** be equal
- If unnormalizedCoordinates is VK_TRUE, mipmapMode **must** be VK_SAMPLER_MIPMAP_MODE_NEAREST
- If unnormalizedCoordinates is VK_TRUE, minLod and maxLod **must** be zero
- 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
- If unnormalizedCoordinates is VK_TRUE, anisotropyEnable **must** be VK_FALSE
- If unnormalizedCoordinates is VK_TRUE, compareEnable **must** be VK_FALSE
- If any of addressModeU, addressModeV or addressModeW are VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER, borderColor **must** be a valid VkBorderColor value
- 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
- If compareEnable is VK_TRUE, compareOp **must** be a valid VkCompareOp value

Valid Usage (Implicit)

- sType **must** be VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO
- pNext **must** be NULL
- flags **must** be 0
- magFilter **must** be a valid VkFilter value
- minFilter **must** be a valid VkFilter value
- mipmapMode **must** be a valid VkSamplerMipmapMode value
- addressModeU **must** be a valid VkSamplerAddressMode value
- addressModeV **must** be a valid VkSamplerAddressMode value
- addressModeW **must** be a valid VkSamplerAddressMode value

See Also

VkBool32, VkBorderColor, VkCompareOp, VkFilter, VkSamplerAddressMode, VkSamplerCreateFlags, VkSamplerMipmapMode, VkStructureType, vkCreateSampler

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSamplerCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSemaphoreCreateInfo(3)

Name

VkSemaphoreCreateInfo - Structure specifying parameters of a newly created semaphore

C Specification

The `VkSemaphoreCreateInfo` structure is defined as:

```
typedef struct VkSemaphoreCreateInfo {
    VkStructureType           sType;
    const void*               pNext;
    VkSemaphoreCreateFlags    flags;
} VkSemaphoreCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.

Description

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`

See Also

VkSemaphoreCreateFlags, VkStructureType, vkCreateSemaphore

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSemaphoreCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkShaderModuleCreateInfo(3)

Name

VkShaderModuleCreateInfo - Structure specifying parameters of a newly created shader module

C Specification

The `VkShaderModuleCreateInfo` structure is defined as:

```
typedef struct VkShaderModuleCreateInfo {
    VkStructureType              sType;
    const void*                  pNext;
    VkShaderModuleCreateFlags    flags;
    size_t                       codeSize;
    const uint32_t*              pCode;
} VkShaderModuleCreateInfo;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific structure.
- `flags` is reserved for future use.
- `codeSize` is the size, in bytes, of the code pointed to by `pCode`.
- `pCode` points to code that is used to create the shader module. The type and format of the code is determined from the content of the memory addressed by `pCode`.

Description

Valid Usage

- `codeSize` **must** be greater than 0
- `codeSize` **must** be a multiple of 4
- `pCode` **must** point to valid SPIR-V code, formatted and packed as described by the Khronos SPIR-V Specification
- `pCode` **must** adhere to the validation rules described by the Validation Rules within a Module section of the SPIR-V Environment appendix
- `pCode` **must** declare the `Shader` capability for 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
- If `pCode` declares any of the capabilities that are listed as not required by the implementation, the relevant feature **must** be enabled, as listed in the SPIR-V Environment appendix

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO`
- `pNext` **must** be `NULL`
- `flags` **must** be `0`
- `pCode` **must** be a pointer to an array of $codeSiz\frac{e}{4}$ `uint32_t` values

See Also

VkShaderModuleCreateFlags, VkStructureType, vkCreateShaderModule

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkShaderModuleCreateInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSparseBufferMemoryBindInfo(3)

Name

VkSparseBufferMemoryBindInfo - Structure specifying a sparse buffer memory bind operation

C Specification

Memory is bound to `VkBuffer` objects created with the `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` flag using the following structure:

```
typedef struct VkSparseBufferMemoryBindInfo {
    VkBuffer                     buffer;
    uint32_t                     bindCount;
    const VkSparseMemoryBind*    pBinds;
} VkSparseBufferMemoryBindInfo;
```

Members

- `buffer` is the `VkBuffer` object to be bound.
- `bindCount` is the number of `VkSparseMemoryBind` structures in the `pBinds` array.
- `pBinds` is a pointer to array of `VkSparseMemoryBind` structures.

Description

Valid Usage (Implicit)

- `buffer` **must** be a valid `VkBuffer` handle
- `pBinds` **must** be a pointer to an array of `bindCount` valid `VkSparseMemoryBind` structures
- `bindCount` **must** be greater than `0`

See Also

VkBindSparseInfo, VkBuffer, VkSparseMemoryBind

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSparseBufferMemoryBindInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSparseImageFormatProperties(3)

Name

VkSparseImageFormatProperties - Structure specifying sparse image format properties

C Specification

The `VkSparseImageFormatProperties` structure is defined as:

```
typedef struct VkSparseImageFormatProperties {
    VkImageAspectFlags          aspectMask;
    VkExtent3D                  imageGranularity;
    VkSparseImageFormatFlags    flags;
} VkSparseImageFormatProperties;
```

Members

- `aspectMask` is a bitmask VkImageAspectFlagBits specifying which aspects of the image the properties apply to.
- `imageGranularity` is the width, height, and depth of the sparse image block in texels or compressed texel blocks.
- `flags` is a bitmask of VkSparseImageFormatFlagBits specifying additional information about the sparse resource.

Description

See Also

VkExtent3D, VkImageAspectFlags, VkSparseImageFormatFlags, VkSparseImageMemoryRequirements, vkGetPhysicalDeviceSparseImageFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSparseImageFormatProperties

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSparseImageMemoryBind(3)

Name

VkSparseImageMemoryBind - Structure specifying sparse image memory bind

C Specification

The `VkSparseImageMemoryBind` structure is defined as:

```
typedef struct VkSparseImageMemoryBind {
    VkImageSubresource        subresource;
    VkOffset3D                offset;
    VkExtent3D                extent;
    VkDeviceMemory            memory;
    VkDeviceSize              memoryOffset;
    VkSparseMemoryBindFlags   flags;
} VkSparseImageMemoryBind;
```

Members

- `subresource` is the aspectMask and region of interest in the image.
- `offset` are the coordinates of the first texel within the image subresource to bind.
- `extent` is the size in texels of the region within the image subresource to bind. The extent **must** be a multiple of the sparse image block dimensions, except when binding sparse image blocks along the edge of an image subresource it **can** instead be such that any coordinate of `offset` + `extent` equals the corresponding dimensions of the image subresource.
- `memory` is the `VkDeviceMemory` object that the sparse image blocks of the image are bound to. If `memory` is VK_NULL_HANDLE, the sparse image blocks are unbound.
- `memoryOffset` is an offset into `VkDeviceMemory` object. If `memory` is VK_NULL_HANDLE, this value is ignored.
- `flags` are sparse memory binding flags.

Description

Valid Usage

- If the sparse aliased residency feature is not enabled, and if any other resources are bound to ranges of `memory`, the range of `memory` being bound **must** not overlap with those bound ranges
- `memory` and `memoryOffset` **must** match the memory requirements of the calling command's `image`, as described in section html/vkspec.html#resources-association
- `subresource` **must** be a valid image subresource for `image` (see html/vkspec.html#resources-image-views)
- `offset.x` **must** be a multiple of the sparse image block width (`VkSparseImageFormatProperties`::`imageGranularity.width`) of the image
- `extent.width` **must** either be a multiple of the sparse image block width of the image, or else (`extent.width` + `offset.x`) **must** equal the width of the image subresource
- `offset.y` **must** be a multiple of the sparse image block height (`VkSparseImageFormatProperties`::`imageGranularity.height`) of the image
- `extent.height` **must** either be a multiple of the sparse image block height of the image, or else (`extent.height` + `offset.y`) **must** equal the height of the image subresource
- `offset.z` **must** be a multiple of the sparse image block depth (`VkSparseImageFormatProperties`::`imageGranularity.depth`) of the image
- `extent.depth` **must** either be a multiple of the sparse image block depth of the image, or else (`extent.depth` + `offset.z`) **must** equal the depth of the image subresource

Valid Usage (Implicit)

- `subresource` **must** be a valid `VkImageSubresource` structure
- If `memory` is not VK_NULL_HANDLE, `memory` **must** be a valid `VkDeviceMemory` handle
- `flags` **must** be a valid combination of VkSparseMemoryBindFlagBits values

See Also

VkDeviceMemory, `VkDeviceSize`, VkExtent3D, VkImageSubresource, VkOffset3D, VkSparseImageMemoryBindInfo, VkSparseMemoryBindFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSparseImageMemoryBind

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSparseImageMemoryBindInfo(3)

Name

VkSparseImageMemoryBindInfo - Structure specifying sparse image memory bind info

C Specification

Memory **can** be bound to sparse image blocks of `VkImage` objects created with the `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT` flag using the following structure:

```
typedef struct VkSparseImageMemoryBindInfo {
    VkImage                           image;
    uint32_t                          bindCount;
    const VkSparseImageMemoryBind*    pBinds;
} VkSparseImageMemoryBindInfo;
```

Members

- `image` is the `VkImage` object to be bound
- `bindCount` is the number of `VkSparseImageMemoryBind` structures in pBinds array
- `pBinds` is a pointer to array of `VkSparseImageMemoryBind` structures

Description

Valid Usage (Implicit)

- `image` **must** be a valid `VkImage` handle
- `pBinds` **must** be a pointer to an array of `bindCount` valid `VkSparseImageMemoryBind` structures
- `bindCount` **must** be greater than `0`

See Also

VkBindSparseInfo, VkImage, VkSparseImageMemoryBind

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSparseImageMemoryBindInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSparseImageMemoryRequirements(3)

Name

VkSparseImageMemoryRequirements - Structure specifying sparse image memory requirements

C Specification

The `VkSparseImageMemoryRequirements` structure is defined as:

```
typedef struct VkSparseImageMemoryRequirements {
    VkSparseImageFormatProperties    formatProperties;
    uint32_t                         imageMipTailFirstLod;
    VkDeviceSize                     imageMipTailSize;
    VkDeviceSize                     imageMipTailOffset;
    VkDeviceSize                     imageMipTailStride;
} VkSparseImageMemoryRequirements;
```

Members

- `formatProperties.aspectMask` is the set of aspects of the image that this sparse memory requirement applies to. This will usually have a single aspect specified. However, depth/stencil images **may** have depth and stencil data interleaved in the same sparse block, in which case both `VK_IMAGE_ASPECT_DEPTH_BIT` and `VK_IMAGE_ASPECT_STENCIL_BIT` would be present.
- `formatProperties.imageGranularity` describes the dimensions of a single bindable sparse image block in pixel units. For aspect `VK_IMAGE_ASPECT_METADATA_BIT`, all dimensions will be zero pixels. All metadata is located in the mip tail region.
- `formatProperties.flags` is a bitmask of VkSparseImageFormatFlagBits:
    - If `VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT` is set the image uses a single mip tail region for all array layers.
    - If `VK_SPARSE_IMAGE_FORMAT_ALIGNED_MIP_SIZE_BIT` is set the dimensions of mip levels **must** be integer multiples of the corresponding dimensions of the sparse image block for levels not located in the mip tail.
    - If `VK_SPARSE_IMAGE_FORMAT_NONSTANDARD_BLOCK_SIZE_BIT` is set the image uses non-standard sparse image block dimensions. The `formatProperties.imageGranularity` values do not match the standard sparse image block dimension corresponding to the image's pixel format.
- `imageMipTailFirstLod` is the first mip level at which image subresources are included in the mip tail region.
- `imageMipTailSize` is the memory size (in bytes) of the mip tail region. If `formatProperties.flags` contains `VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT`, this is the size of the whole mip tail, otherwise this is the size of the mip tail of a single array layer. This value is guaranteed to be a multiple of the sparse block size in bytes.
- `imageMipTailOffset` is the opaque memory offset used with VkSparseImageOpaqueMemoryBindInfo to bind the mip tail region(s).

- `imageMipTailStride` is the offset stride between each array-layer's mip tail, if `formatProperties.flags` does not contain `VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT` (otherwise the value is undefined).

Description

See Also

`VkDeviceSize`, VkSparseImageFormatProperties, vkGetImageSparseMemoryRequirements

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSparseImageMemoryRequirements

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSparseImageOpaqueMemoryBindInfo(3)

Name

VkSparseImageOpaqueMemoryBindInfo - Structure specifying sparse image opaque memory bind info

C Specification

Memory is bound to opaque regions of `VkImage` objects created with the `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` flag using the following structure:

```
typedef struct VkSparseImageOpaqueMemoryBindInfo {
    VkImage                      image;
    uint32_t                     bindCount;
    const VkSparseMemoryBind*    pBinds;
} VkSparseImageOpaqueMemoryBindInfo;
```

Members

- `image` is the `VkImage` object to be bound.
- `bindCount` is the number of `VkSparseMemoryBind` structures in the `pBinds` array.
- `pBinds` is a pointer to array of `VkSparseMemoryBind` structures.

Description

Valid Usage

- For any given element of `pBinds`, if the `flags` member of that element contains `VK_SPARSE_MEMORY_BIND_METADATA_BIT`, the binding range defined **must** be within the mip tail region of the metadata aspect of `image`

Valid Usage (Implicit)

- `image` **must** be a valid `VkImage` handle
- `pBinds` **must** be a pointer to an array of `bindCount` valid `VkSparseMemoryBind` structures
- `bindCount` **must** be greater than `0`

See Also

VkBindSparseInfo, VkImage, VkSparseMemoryBind

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSparseImageOpaqueMemoryBindInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSparseMemoryBind(3)

Name

VkSparseMemoryBind - Structure specifying a sparse memory bind operation

C Specification

The `VkSparseMemoryBind` structure is defined as:

```
typedef struct VkSparseMemoryBind {
    VkDeviceSize               resourceOffset;
    VkDeviceSize               size;
    VkDeviceMemory             memory;
    VkDeviceSize               memoryOffset;
    VkSparseMemoryBindFlags    flags;
} VkSparseMemoryBind;
```

Members

- `resourceOffset` is the offset into the resource.
- `size` is the size of the memory region to be bound.
- `memory` is the `VkDeviceMemory` object that the range of the resource is bound to. If `memory` is VK_NULL_HANDLE, the range is unbound.
- `memoryOffset` is the offset into the `VkDeviceMemory` object to bind the resource range to. If `memory` is VK_NULL_HANDLE, this value is ignored.
- `flags` is a bitmask of VkSparseMemoryBindFlagBits specifying usage of the binding operation.

Description

The *binding range* [`resourceOffset`, `resourceOffset` + `size`) has different constraints based on `flags`. If `flags` contains `VK_SPARSE_MEMORY_BIND_METADATA_BIT`, the binding range **must** be within the mip tail region of the metadata aspect. This metadata region is defined by:

metadataRegion = [base, base + `imageMipTailSize`)

base = `imageMipTailOffset` + `imageMipTailStride` × n

and `imageMipTailOffset`, `imageMipTailSize`, and `imageMipTailStride` values are from the VkSparseImageMemoryRequirements corresponding to the metadata aspect of the image, and n is a valid array layer index for the image,

`imageMipTailStride` is considered to be zero for aspects where `VkSparseImageMemoryRequirements` ::`formatProperties.flags` contains `VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT`.

If `flags` does not contain `VK_SPARSE_MEMORY_BIND_METADATA_BIT`, the binding range **must** be within the range [0,VkMemoryRequirements::size).

Valid Usage

- If memory is not VK_NULL_HANDLE, memory and memoryOffset **must** match the memory requirements of the resource, as described in section html/vkspec.html#resources-association
- If memory is not VK_NULL_HANDLE, memory **must** not have been created with a memory type that reports VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT bit set
- size **must** be greater than 0
- resourceOffset **must** be less than the size of the resource
- size **must** be less than or equal to the size of the resource minus resourceOffset
- memoryOffset **must** be less than the size of memory
- size **must** be less than or equal to the size of memory minus memoryOffset

Valid Usage (Implicit)

- If memory is not VK_NULL_HANDLE, memory **must** be a valid VkDeviceMemory handle
- flags **must** be a valid combination of VkSparseMemoryBindFlagBits values

See Also

VkDeviceMemory, VkDeviceSize, VkSparseBufferMemoryBindInfo, VkSparseImageOpaqueMemoryBindInfo, VkSparseMemoryBindFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSparseMemoryBind

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSpecializationInfo(3)

Name

VkSpecializationInfo - Structure specifying specialization info

C Specification

The `VkSpecializationInfo` structure is defined as:

```
typedef struct VkSpecializationInfo {
    uint32_t                           mapEntryCount;
    const VkSpecializationMapEntry*    pMapEntries;
    size_t                             dataSize;
    const void*                        pData;
} VkSpecializationInfo;
```

Members

- `mapEntryCount` is the number of entries in the `pMapEntries` array.
- `pMapEntries` is a pointer to an array of `VkSpecializationMapEntry` which maps constant IDs to offsets in `pData`.
- `dataSize` is the byte size of the `pData` buffer.
- `pData` contains the actual constant values to specialize with.

Description

`pMapEntries` points to a structure of type VkSpecializationMapEntry.

Valid Usage

- The `offset` member of any given element of `pMapEntries` **must** be less than `dataSize`
- For any given element of `pMapEntries`, `size` **must** be less than or equal to `dataSize` minus `offset`
- If `mapEntryCount` is not `0`, `pMapEntries` **must** be a pointer to an array of `mapEntryCount` valid `VkSpecializationMapEntry` structures

Valid Usage (Implicit)

- If `dataSize` is not `0`, `pData` **must** be a pointer to an array of `dataSize` bytes

See Also

VkPipelineShaderStageCreateInfo, VkSpecializationMapEntry

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSpecializationInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSpecializationMapEntry(3)

Name

VkSpecializationMapEntry - Structure specifying a specialization map entry

C Specification

The `VkSpecializationMapEntry` structure is defined as:

```
typedef struct VkSpecializationMapEntry {
    uint32_t    constantID;
    uint32_t    offset;
    size_t      size;
} VkSpecializationMapEntry;
```

Members

- `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.

Description

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

- 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`

See Also

VkSpecializationInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSpecializationMapEntry

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkStencilOpState(3)

Name

VkStencilOpState - Structure specifying stencil operation state

C Specification

The `VkStencilOpState` structure is defined as:

```
typedef struct VkStencilOpState {
    VkStencilOp    failOp;
    VkStencilOp    passOp;
    VkStencilOp    depthFailOp;
    VkCompareOp    compareOp;
    uint32_t       compareMask;
    uint32_t       writeMask;
    uint32_t       reference;
} VkStencilOpState;
```

Members

- `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 reference value that is used in the unsigned stencil comparison.

Description

Valid Usage (Implicit)

- `failOp` **must** be a valid VkStencilOp value
- `passOp` **must** be a valid VkStencilOp value
- `depthFailOp` **must** be a valid VkStencilOp value
- `compareOp` **must** be a valid VkCompareOp value

See Also

VkCompareOp, VkPipelineDepthStencilStateCreateInfo, VkStencilOp

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkStencilOpState

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSubmitInfo(3)

Name

VkSubmitInfo - Structure specifying a queue submit operation

C Specification

The VkSubmitInfo structure is defined as:

```
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;
```

Members

- sType is the type of this structure.
- pNext is NULL or a pointer to an extension-specific 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 semaphores 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 command buffers 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 semaphores 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.

Description

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

- Any given element of `pCommandBuffers` **must** not have been allocated with `VK_COMMAND_BUFFER_LEVEL_SECONDARY`
- If the geometry shaders feature is not enabled, any given element of `pWaitDstStageMask` **must** not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
- If the tessellation shaders feature is not enabled, any given element of `pWaitDstStageMask` **must** not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
- Any given element of `pWaitDstStageMask` **must** not include `VK_PIPELINE_STAGE_HOST_BIT`.

Valid Usage (Implicit)

- `sType` **must** be `VK_STRUCTURE_TYPE_SUBMIT_INFO`
- `pNext` **must** be `NULL`
- If `waitSemaphoreCount` is not `0`, `pWaitSemaphores` **must** be a pointer to an array of `waitSemaphoreCount` valid `VkSemaphore` handles
- If `waitSemaphoreCount` is not `0`, `pWaitDstStageMask` **must** be a pointer to an array of `waitSemaphoreCount` valid combinations of VkPipelineStageFlagBits values
- Each element of `pWaitDstStageMask` **must** not be `0`
- If `commandBufferCount` is not `0`, `pCommandBuffers` **must** be a pointer to an array of `commandBufferCount` valid `VkCommandBuffer` handles
- If `signalSemaphoreCount` is not `0`, `pSignalSemaphores` **must** be a pointer to an array of `signalSemaphoreCount` valid `VkSemaphore` handles
- Each of the elements of `pCommandBuffers`, the elements of `pSignalSemaphores`, and the elements of `pWaitSemaphores` that are valid handles **must** have been created, allocated, or retrieved from the same `VkDevice`

See Also

VkCommandBuffer, VkPipelineStageFlags, VkSemaphore, VkStructureType, vkQueueSubmit

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSubmitInfo

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the

Specification, not directly.

VkSubpassDependency(3)

Name

VkSubpassDependency - Structure specifying a subpass dependency

C Specification

The `VkSubpassDependency` structure is defined as:

```
typedef struct VkSubpassDependency {
    uint32_t                srcSubpass;
    uint32_t                dstSubpass;
    VkPipelineStageFlags    srcStageMask;
    VkPipelineStageFlags    dstStageMask;
    VkAccessFlags           srcAccessMask;
    VkAccessFlags           dstAccessMask;
    VkDependencyFlags       dependencyFlags;
} VkSubpassDependency;
```

Members

- `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.
- `dstStageMask` is a bitmask of VkPipelineStageFlagBits specifying the destination stage mask
- `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.

Description

If `srcSubpass` is equal to `dstSubpass` then the VkSubpassDependency describes a subpass self-dependency, and only constrains the pipeline barriers allowed within a subpass instance. Otherwise, when a render pass instance which includes a subpass dependency is submitted to a queue, it defines a memory dependency between the subpasses identified by `srcSubpass` and `dstSubpass`.

If `srcSubpass` is equal to `VK_SUBPASS_EXTERNAL`, the first synchronization scope includes commands submitted to the queue before the render pass instance began. 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 submitted after the render pass instance is ended. Otherwise, the second set of commands includes all commands submitted as part of the subpass instance identified by `dstSubpass` and any load, store or 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 access 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 access 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.

Valid Usage

- If srcSubpass is not VK_SUBPASS_EXTERNAL, srcStageMask **must** not include VK_PIPELINE_STAGE_HOST_BIT
- If dstSubpass is not VK_SUBPASS_EXTERNAL, dstStageMask **must** not include VK_PIPELINE_STAGE_HOST_BIT
- If the geometry shaders feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- If the geometry shaders feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- If the tessellation shaders feature is not enabled, srcStageMask **must** not contain VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- If the tessellation shaders feature is not enabled, dstStageMask **must** not contain VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT or VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- 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
- srcSubpass and dstSubpass **must** not both be equal to VK_SUBPASS_EXTERNAL
- If srcSubpass is equal to dstSubpass, srcStageMask and dstStageMask **must** only contain one of VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT, 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_BOTTOM_OF_PIPE_BIT, or VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
- If srcSubpass is equal to dstSubpass and not all of the stages in srcStageMask and dstStageMask are framebuffer-space stages, the logically latest pipeline stage in srcStageMask **must** be logically earlier than or equal to the logically earliest pipeline stage in dstStageMask
- 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.
- 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.

Valid Usage (Implicit)

- `srcStageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- `srcStageMask` **must** not be `0`
- `dstStageMask` **must** be a valid combination of VkPipelineStageFlagBits values
- `dstStageMask` **must** not be `0`
- `srcAccessMask` **must** be a valid combination of VkAccessFlagBits values
- `dstAccessMask` **must** be a valid combination of VkAccessFlagBits values
- `dependencyFlags` **must** be a valid combination of VkDependencyFlagBits values

See Also

VkAccessFlags, VkDependencyFlags, VkPipelineStageFlags, VkRenderPassCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSubpassDependency

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSubpassDescription(3)

Name

VkSubpassDescription - Structure specifying a subpass description

C Specification

The `VkSubpassDescription` structure is defined as:

```
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;
```

Members

- `flags` is a bitmask of VkSubpassDescriptionFlagBits specifying usage of the subpass.
- `pipelineBindPoint` is a VkPipelineBindPoint value specifying whether this is a compute or graphics subpass. Currently, only graphics subpasses are supported.
- `inputAttachmentCount` is the number of input attachments.
- `pInputAttachments` is an array of VkAttachmentReference structures (defined below) that lists which of the render pass's attachments **can** be read in the shader during the subpass, and what layout each attachment will be in during the subpass. Each element of the array corresponds to an input attachment unit number in the shader, i.e. if the shader declares an input variable `layout(input_attachment_index=X, set=Y, binding=Z)` then it uses the attachment provided in `pInputAttachments`[X]. Input attachments **must** also be bound to the pipeline with a descriptor set, with the input attachment descriptor written in the location (set=Y, binding=Z).
- `colorAttachmentCount` is the number of color attachments.
- `pColorAttachments` is an array of `colorAttachmentCount` VkAttachmentReference structures that lists which of the render pass's attachments will be used as color attachments in the subpass, and what layout each attachment will be in during the subpass. Each element of the array corresponds to a fragment shader output location, i.e. if the shader declared an output variable `layout(location=X)` then it uses the attachment provided in `pColorAttachments`[X].
- `pResolveAttachments` is `NULL` or an array of `colorAttachmentCount` VkAttachmentReference structures that lists which of the render pass's attachments are resolved to at the end of the subpass, and what layout each attachment will be in during the multisample resolve operation.

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. At the end of each subpass, multisample resolve operations read the subpass's color attachments, and resolve the samples for each pixel to the same pixel location in the corresponding resolve attachments, unless the resolve attachment index is `VK_ATTACHMENT_UNUSED`. If the first use of an attachment in a render pass is as a resolve attachment, then the `loadOp` is effectively ignored as the resolve is guaranteed to overwrite all pixels in the render area.

- `pDepthStencilAttachment` is a pointer to a VkAttachmentReference specifying which attachment will be used for depth/stencil data and the layout it will be in during the subpass. Setting the attachment index to `VK_ATTACHMENT_UNUSED` or leaving this pointer as `NULL` indicates that no depth/stencil attachment will be used in the subpass.
- `preserveAttachmentCount` is the number of preserved attachments.
- `pPreserveAttachments` is an array of `preserveAttachmentCount` render pass attachment indices describing the attachments that are not used by a subpass, but whose contents **must** be preserved throughout the subpass.

Description

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 $\mathbf{S_1}$ that uses or preserves the attachment, and a subpass dependency from $\mathbf{S_1}$ to **S**.
- The attachment is not used or preserved in subpass **S**.

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 $\mathbf{S_1}$ if those subpasses use or preserve the attachment.

Valid Usage

- `pipelineBindPoint` **must** be `VK_PIPELINE_BIND_POINT_GRAPHICS`
- `colorAttachmentCount` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxColorAttachments`
- 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`
- 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`
- If `pResolveAttachments` is not `NULL`, the sample count of each element of `pColorAttachments` **must** be anything other than `VK_SAMPLE_COUNT_1_BIT`
- Any given element of `pResolveAttachments` **must** have a sample count of `VK_SAMPLE_COUNT_1_BIT`
- Any given element of `pResolveAttachments` **must** have the same VkFormat as its corresponding color attachment
- All attachments in `pColorAttachments` that are not `VK_ATTACHMENT_UNUSED` **must** have the same sample count
- If `pDepthStencilAttachment` is not `VK_ATTACHMENT_UNUSED` and any attachments in `pColorAttachments` are not `VK_ATTACHMENT_UNUSED`, they **must** have the same sample count
- If any input attachments are `VK_ATTACHMENT_UNUSED`, then any pipelines bound during the subpass **must** not access those input attachments from the fragment shader
- The `attachment` member of any element of `pPreserveAttachments` **must** not be `VK_ATTACHMENT_UNUSED`
- Any given element of `pPreserveAttachments` **must** not also be an element of any other member of the subpass description
- If any attachment is used as both an input attachment and a color or depth/stencil attachment, then each use **must** use the same `layout`

Valid Usage (Implicit)

- `flags` **must** be a valid combination of VkSubpassDescriptionFlagBits values
- `pipelineBindPoint` **must** be a valid VkPipelineBindPoint value
- If `inputAttachmentCount` is not `0`, `pInputAttachments` **must** be a pointer to an array of `inputAttachmentCount` valid `VkAttachmentReference` structures
- If `colorAttachmentCount` is not `0`, `pColorAttachments` **must** be a pointer to an array of `colorAttachmentCount` valid `VkAttachmentReference` structures
- If `colorAttachmentCount` is not `0`, and `pResolveAttachments` is not `NULL`, `pResolveAttachments` **must** be a pointer to an array of `colorAttachmentCount` valid `VkAttachmentReference` structures
- If `pDepthStencilAttachment` is not `NULL`, `pDepthStencilAttachment` **must** be a pointer to a valid `VkAttachmentReference` structure
- If `preserveAttachmentCount` is not `0`, `pPreserveAttachments` **must** be a pointer to an array of `preserveAttachmentCount` `uint32_t` values

See Also

VkAttachmentReference, VkPipelineBindPoint, VkRenderPassCreateInfo, VkSubpassDescriptionFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSubpassDescription

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSubresourceLayout(3)

Name

VkSubresourceLayout - Structure specifying subresource layout

C Specification

Information about the layout of the image subresource is returned in a `VkSubresourceLayout` structure:

```
typedef struct VkSubresourceLayout {
    VkDeviceSize    offset;
    VkDeviceSize    size;
    VkDeviceSize    rowPitch;
    VkDeviceSize    arrayPitch;
    VkDeviceSize    depthPitch;
} VkSubresourceLayout;
```

Members

- `offset` is the byte offset from the start of the image 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.

Description

For images created with linear tiling, `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;
```

arrayPitch is undefined for images that were not created as arrays. depthPitch is defined only for 3D images.

For color formats, the aspectMask member of VkImageSubresource **must** be VK_IMAGE_ASPECT_COLOR_BIT. For depth/stencil formats, 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.

See Also

VkDeviceSize, vkGetImageSubresourceLayout

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSubresourceLayout

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkVertexInputAttributeDescription(3)

Name

VkVertexInputAttributeDescription - Structure specifying vertex input attribute description

C Specification

Each vertex input attribute is specified by an instance of the `VkVertexInputAttributeDescription` structure.

The `VkVertexInputAttributeDescription` structure is defined as:

```
typedef struct VkVertexInputAttributeDescription {
    uint32_t    location;
    uint32_t    binding;
    VkFormat    format;
    uint32_t    offset;
} VkVertexInputAttributeDescription;
```

Members

- `location` is the shader binding 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.

Description

Valid Usage

- `location` **must** be less than `VkPhysicalDeviceLimits`::`maxVertexInputAttributes`
- `binding` **must** be less than `VkPhysicalDeviceLimits`::`maxVertexInputBindings`
- `offset` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxVertexInputAttributeOffset`
- `format` **must** be allowed as a vertex buffer format, as specified by the `VK_FORMAT_FEATURE_VERTEX_BUFFER_BIT` flag in `VkFormatProperties`::`bufferFeatures` returned by `vkGetPhysicalDeviceFormatProperties`

Valid Usage (Implicit)

- `format` **must** be a valid VkFormat value

See Also

VkFormat, VkPipelineVertexInputStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkVertexInputAttributeDescription

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkVertexInputBindingDescription(3)

Name

VkVertexInputBindingDescription - Structure specifying vertex input binding description

C Specification

The `VkVertexInputBindingDescription` structure is defined as:

```
typedef struct VkVertexInputBindingDescription {
    uint32_t             binding;
    uint32_t             stride;
    VkVertexInputRate    inputRate;
} VkVertexInputBindingDescription;
```

Members

- `binding` is the binding number that this structure describes.
- `stride` is the distance in bytes between two 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.

Description

Valid Usage

- `binding` **must** be less than `VkPhysicalDeviceLimits`::`maxVertexInputBindings`
- `stride` **must** be less than or equal to `VkPhysicalDeviceLimits`::`maxVertexInputBindingStride`

Valid Usage (Implicit)

- `inputRate` **must** be a valid VkVertexInputRate value

See Also

VkPipelineVertexInputStateCreateInfo, VkVertexInputRate

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkVertexInputBindingDescription

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkViewport(3)

Name

VkViewport - Structure specifying a viewport

C Specification

The `VkViewport` structure is defined as:

```
typedef struct VkViewport {
    float    x;
    float    y;
    float    width;
    float    height;
    float    minDepth;
    float    maxDepth;
} VkViewport;
```

Members

- `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. It is valid for `minDepth` to be greater than or equal to `maxDepth`.

Description

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 ∈ { 0, 1, ..., 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` + `width` / 2

o_y = `y` + `height` / 2

o_z = `minDepth`

p_x = `width`

p_y = `height`

p_z = `maxDepth` - `minDepth`.

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

- `width` **must** be greater than `0.0` and less than or equal to `VkPhysicalDeviceLimits` ::`maxViewportDimensions`[0]
- `height` **must** be greater than `0.0` and less than or equal to `VkPhysicalDeviceLimits` ::`maxViewportDimensions`[1]
- `x` and `y` **must** each be between `viewportBoundsRange`[0] and `viewportBoundsRange`[1], inclusive
- (`x` + `width`) **must** be less than or equal to `viewportBoundsRange`[1]
- (`y` + `height`) **must** be less than or equal to `viewportBoundsRange`[1]
- `minDepth` **must** be between `0.0` and `1.0`, inclusive
- `maxDepth` **must** be between `0.0` and `1.0`, inclusive

See Also

VkPipelineViewportStateCreateInfo, vkCmdSetViewport

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkViewport

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkWriteDescriptorSet(3)

Name

VkWriteDescriptorSet - Structure specifying the parameters of a descriptor set write operation

C Specification

The `VkWriteDescriptorSet` structure is defined as:

```
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;
```

Members

- `sType` is the type of this structure.
- `pNext` is `NULL` or a pointer to an extension-specific 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 (the number of elements in `pImageInfo`, `pBufferInfo`, or `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 that specified in `VkDescriptorSetLayoutBinding` for `dstSet` at `dstBinding`. The type of the descriptor also controls which array the descriptors are taken from.
- `pImageInfo` points to an array of VkDescriptorImageInfo structures or is ignored, as described below.
- `pBufferInfo` points to an array of VkDescriptorBufferInfo structures or is ignored, as described below.
- `pTexelBufferView` points to an array of VkBufferView handles as described in the Buffer Views section or is ignored, as described below.

Description

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 `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.

Valid Usage

- 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
- dstBinding **must** be a binding with a non-zero descriptorCount
- All consecutive bindings updated via a single VkWriteDescriptorSet structure, except those with a descriptorCount of zero, **must** have identical descriptorType and stageFlags.
- 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.
- descriptorType **must** match the type of dstBinding within dstSet
- dstSet **must** be a valid VkDescriptorSet handle
- 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, as described by html/vkspec.html#descriptorsets-updates-consecutive
- If 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, pImageInfo **must** be a pointer to an array of descriptorCount valid VkDescriptorImageInfo structures
- If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, pTexelBufferView **must** be a pointer to an array of descriptorCount valid VkBufferView handles
- 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 pointer to an array of descriptorCount valid VkDescriptorBufferInfo structures
- 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 any given element of pImageInfo **must** be a valid VkSampler object
- If descriptorType is VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, the imageView and imageLayout members of any given element of pImageInfo **must** be a valid VkImageView and VkImageLayout, respectively
- If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, for each descriptor that will be accessed via load or store operations the imageLayout member for corresponding elements of pImageInfo **must** be VK_IMAGE_LAYOUT_GENERAL
- If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, the offset member of any given element of pBufferInfo **must** be a multiple of VkPhysicalDeviceLimits

::minUniformBufferOffsetAlignment

- If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, the offset member of any given element of pBufferInfo **must** be a multiple of VkPhysicalDeviceLimits ::minStorageBufferOffsetAlignment
- 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, the buffer member of any given element of pBufferInfo that is non-sparse **must** be bound completely and contiguously to a single VkDeviceMemory object
- If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, the buffer member of any given element of pBufferInfo **must** have been created with VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT set
- If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, the buffer member of any given element of pBufferInfo **must** have been created with VK_BUFFER_USAGE_STORAGE_BUFFER_BIT set
- If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, the range member of any given element of pBufferInfo, or the effective range if range is VK_WHOLE_SIZE, **must** be less than or equal to VkPhysicalDeviceLimits::maxUniformBufferRange
- If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, the range member of any given element of pBufferInfo, or the effective range if range is VK_WHOLE_SIZE, **must** be less than or equal to VkPhysicalDeviceLimits::maxStorageBufferRange
- If descriptorType is VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, the VkBuffer that any given element of pTexelBufferView was created from **must** have been created with VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT set
- If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, the VkBuffer that any given element of pTexelBufferView was created from **must** have been created with VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT set
- If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_IMAGE or VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, the imageView member of any given element of pImageInfo **must** have been created with the identity swizzle
- If descriptorType is VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, the imageView member of any given element of pImageInfo **must** have been created with VK_IMAGE_USAGE_SAMPLED_BIT set
- If descriptorType is VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, the imageLayout member of any given element of pImageInfo **must** be VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL or VK_IMAGE_LAYOUT_GENERAL
- If descriptorType is VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, the imageView member of any given element of pImageInfo **must** have been created with VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT set

* If descriptorType is VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, the imageView member of any given element of pImageInfo **must** have been created with VK_IMAGE_USAGE_STORAGE_BIT set

Valid Usage (Implicit)

* sType **must** be VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET
* pNext **must** be NULL
* descriptorType **must** be a valid VkDescriptorType value
* descriptorCount **must** be greater than 0
* Both of dstSet, and the elements of pTexelBufferView that are valid handles **must** have been created, allocated, or retrieved from the same VkDevice

See Also

VkBufferView, VkDescriptorBufferInfo, VkDescriptorImageInfo, VkDescriptorSet, VkDescriptorType, VkStructureType, vkUpdateDescriptorSets

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkWriteDescriptorSet

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

Enumerations

VkAccessFlagBits(3)

Name

VkAccessFlagBits - Bitmask specifying memory access types that will participate in a memory dependency

C Specification

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. Each access type corresponds to a bit flag in VkAccessFlagBits.

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.

Access types that **can** be set in an access mask include:

```
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,
} VkAccessFlagBits;
```

Description

- `VK_ACCESS_INDIRECT_COMMAND_READ_BIT` specifies read access to an indirect command structure read as part of an indirect drawing or dispatch command.

- VK_ACCESS_INDEX_READ_BIT specifies read access to an index buffer as part of an indexed drawing command, bound by vkCmdBindIndexBuffer.
- VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT specifies read access to a vertex buffer as part of a drawing command, bound by vkCmdBindVertexBuffers.
- VK_ACCESS_UNIFORM_READ_BIT specifies read access to a uniform buffer.
- VK_ACCESS_INPUT_ATTACHMENT_READ_BIT specifies read access to an input attachment within a renderpass during fragment shading.
- VK_ACCESS_SHADER_READ_BIT specifies read access to a storage buffer, uniform texel buffer, storage texel buffer, sampled image, or storage image.
- VK_ACCESS_SHADER_WRITE_BIT specifies write access to a storage buffer, storage texel buffer, or storage image.
- VK_ACCESS_COLOR_ATTACHMENT_READ_BIT specifies read access to a color attachment, such as via blending, logic operations, or via certain subpass load operations.
- VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT specifies write access to a color or resolve attachment during a render pass or via certain subpass load and store operations.
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT specifies read access to a depth/stencil attachment, via depth or stencil operations or via certain subpass load operations.
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT specifies write access to a depth/stencil attachment, via depth or stencil operations or via certain subpass load and store operations.
- VK_ACCESS_TRANSFER_READ_BIT specifies read access to an image or buffer in a copy operation.
- VK_ACCESS_TRANSFER_WRITE_BIT specifies write access to an image or buffer in a clear or copy operation.
- 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.
- 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.
- VK_ACCESS_MEMORY_READ_BIT specifies read access via non-specific entities. These entities include the Vulkan device and host, but **may** also include entities external to the Vulkan device or otherwise not part of the core Vulkan pipeline. When included in a destination access mask, makes all available writes visible to all future read accesses on entities known to the Vulkan device.
- VK_ACCESS_MEMORY_WRITE_BIT specifies write access via non-specific entities. These entities include the Vulkan device and host, but **may** also include entities external to the Vulkan device or otherwise not part of the core Vulkan pipeline. When included in a source access mask, all writes that are performed by entities known to the Vulkan device are made available. When included in a destination access mask, makes all available writes visible to all future write accesses on entities known to the Vulkan device.

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 7. Supported access types

Access flag	Supported pipeline stages
VK_ACCESS_INDIRECT_COMMAND_READ_BIT	VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
VK_ACCESS_INDEX_READ_BIT	VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT	VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
VK_ACCESS_UNIFORM_READ_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, or VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
VK_ACCESS_INPUT_ATTACHMENT_READ_BIT	VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
VK_ACCESS_SHADER_READ_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, or VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
VK_ACCESS_SHADER_WRITE_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, or VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
VK_ACCESS_COLOR_ATTACHMENT_READ_BIT	VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT	VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT	VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT, or VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT	VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT, or VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
VK_ACCESS_TRANSFER_READ_BIT	VK_PIPELINE_STAGE_TRANSFER_BIT
VK_ACCESS_TRANSFER_WRITE_BIT	VK_PIPELINE_STAGE_TRANSFER_BIT
VK_ACCESS_HOST_READ_BIT	VK_PIPELINE_STAGE_HOST_BIT
VK_ACCESS_HOST_WRITE_BIT	VK_PIPELINE_STAGE_HOST_BIT
VK_ACCESS_MEMORY_READ_BIT	N/A
VK_ACCESS_MEMORY_WRITE_BIT	N/A

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 visible to the VK_ACCESS_HOST_WRITE_BIT access type, where it **can** be further made available to the device by synchronization commands. Similarly, vkInvalidateMappedMemoryRanges **must** be called to guarantee that writes which are visible to the VK_ACCESS_HOST_READ_BIT access type 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 visible to the VK_ACCESS_HOST_WRITE_BIT access type. Similarly, writes made visible to the VK_ACCESS_HOST_READ_BIT access type are automatically made visible to the host.

> *Note*
>
> The vkQueueSubmit command automatically guarantees that host writes flushed to VK_ACCESS_HOST_WRITE_BIT are made available if they were flushed before the command executed, 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.

See Also

VkAccessFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkAccessFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkAttachmentDescriptionFlagBits(3)

Name

VkAttachmentDescriptionFlagBits - Bitmask specifying additional properties of an attachment

C Specification

Bits which **can** be set in VkAttachmentDescription::`flags` describing additional properties of the attachment are:

```
typedef enum VkAttachmentDescriptionFlagBits {
    VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT = 0x00000001,
} VkAttachmentDescriptionFlagBits;
```

Description

- `VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT` specifies that the attachment aliases the same device memory as other attachments.

See Also

VkAttachmentDescriptionFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkAttachmentDescriptionFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkAttachmentLoadOp(3)

Name

VkAttachmentLoadOp - Specify how contents of an attachment are treated at the beginning of a subpass

C Specification

Possible values of VkAttachmentDescription::loadOp and stencilLoadOp, specifying how the contents of the attachment are treated, are:

```
typedef enum VkAttachmentLoadOp {
    VK_ATTACHMENT_LOAD_OP_LOAD = 0,
    VK_ATTACHMENT_LOAD_OP_CLEAR = 1,
    VK_ATTACHMENT_LOAD_OP_DONT_CARE = 2,
} VkAttachmentLoadOp;
```

Description

- VK_ATTACHMENT_LOAD_OP_LOAD specifies that the previous contents of the image within the render area will be preserved. 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.

See Also

VkAttachmentDescription

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkAttachmentLoadOp

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkAttachmentStoreOp(3)

Name

VkAttachmentStoreOp - Specify how contents of an attachment are treated at the end of a subpass

C Specification

Possible values of VkAttachmentDescription::`storeOp` and `stencilStoreOp`, specifying how the contents of the attachment are treated, are:

```
typedef enum VkAttachmentStoreOp {
    VK_ATTACHMENT_STORE_OP_STORE = 0,
    VK_ATTACHMENT_STORE_OP_DONT_CARE = 1,
} VkAttachmentStoreOp;
```

Description

- `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`.

See Also

VkAttachmentDescription

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkAttachmentStoreOp

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkBlendFactor(3)

Name

VkBlendFactor - Framebuffer blending factors

C Specification

The source and destination color and alpha blending factors are selected from the enum:

```
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;
```

Description

The semantics of each enum value is described in the table below:

Table 8. 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

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_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.

See Also

VkPipelineColorBlendAttachmentState

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBlendFactor

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkBlendOp(3)

Name

VkBlendOp - Framebuffer blending operations

C Specification

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:

```
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,
} VkBlendOp;
```

Description

The semantics of each basic blend operations is described in the table below:

Table 9. 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.

See Also

VkPipelineColorBlendAttachmentState

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBlendOp

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkBorderColor(3)

Name

VkBorderColor - Specify border color used for texture lookups

C Specification

Possible values of VkSamplerCreateInfo::`borderColor`, specifying the border color used for texture lookups, are:

```
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,
} VkBorderColor;
```

Description

- `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.

These colors are described in detail in Texel Replacement.

See Also

VkSamplerCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBorderColor

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkBufferCreateFlagBits(3)

Name

VkBufferCreateFlagBits - Bitmask specifying additional parameters of a buffer

C Specification

Bits which **can** be set in VkBufferCreateInfo::`flags`, specifying additional parameters of a buffer, are:

```
typedef enum VkBufferCreateFlagBits {
    VK_BUFFER_CREATE_SPARSE_BINDING_BIT = 0x00000001,
    VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT = 0x00000002,
    VK_BUFFER_CREATE_SPARSE_ALIASED_BIT = 0x00000004,
} VkBufferCreateFlagBits;
```

Description

- `VK_BUFFER_CREATE_SPARSE_BINDING_BIT` specifies that the buffer will be backed using sparse memory binding.
- `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.
- `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.

See Sparse Resource Features and Physical Device Features for details of the sparse memory features supported on a device.

See Also

VkBufferCreateFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBufferCreateFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkBufferUsageFlagBits(3)

Name

VkBufferUsageFlagBits - Bitmask specifying allowed usage of a buffer

C Specification

Bits which **can** be set in VkBufferCreateInfo::usage, specifying usage behavior of a buffer, are:

```
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,
} VkBufferUsageFlagBits;
```

Description

- 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`.

See Also

VkBufferUsageFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBufferUsageFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkColorComponentFlagBits(3)

Name

VkColorComponentFlagBits - Bitmask controlling which components are written to the framebuffer

C Specification

Bits which **can** be set in VkPipelineColorBlendAttachmentState::colorWriteMask to determine whether the final color values R, G, B and A are written to the framebuffer attachment are:

```
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;
```

Description

- `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.

See Also

VkColorComponentFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkColorComponentFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCommandBufferLevel(3)

Name

VkCommandBufferLevel - Enumerant specifying a command buffer level

C Specification

Possible values of VkCommandBufferAllocateInfo::`flags`, specifying the command buffer level, are:

```
typedef enum VkCommandBufferLevel {
    VK_COMMAND_BUFFER_LEVEL_PRIMARY = 0,
    VK_COMMAND_BUFFER_LEVEL_SECONDARY = 1,
} VkCommandBufferLevel;
```

Description

- `VK_COMMAND_BUFFER_LEVEL_PRIMARY` specifies a primary command buffer.
- `VK_COMMAND_BUFFER_LEVEL_SECONDARY` specifies a secondary command buffer.

See Also

VkCommandBufferAllocateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandBufferLevel

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCommandBufferResetFlagBits(3)

Name

VkCommandBufferResetFlagBits - Bitmask controlling behavior of a command buffer reset

C Specification

Bits which **can** be set in vkResetCommandBuffer::`flags` to control the reset operation are:

```
typedef enum VkCommandBufferResetFlagBits {
    VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT = 0x00000001,
} VkCommandBufferResetFlagBits;
```

Description

- `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.

See Also

VkCommandBufferResetFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandBufferResetFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCommandBufferUsageFlagBits(3)

Name

VkCommandBufferUsageFlagBits - Bitmask specifying usage behavior for command buffer

C Specification

Bits which **can** be set in VkCommandBufferBeginInfo::`flags` to specify usage behavior for a command buffer are:

```
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;
```

Description

- `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 a queue while it is in the *pending state*, and recorded into multiple primary command buffers.

See Also

VkCommandBufferUsageFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandBufferUsageFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCommandPoolCreateFlagBits(3)

Name

VkCommandPoolCreateFlagBits - Bitmask specifying usage behavior for a command pool

C Specification

Bits which **can** be set in VkCommandPoolCreateInfo::flags to specify usage behavior for a command pool are:

```
typedef enum VkCommandPoolCreateFlagBits {
    VK_COMMAND_POOL_CREATE_TRANSIENT_BIT = 0x00000001,
    VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT = 0x00000002,
} VkCommandPoolCreateFlagBits;
```

Description

- VK_COMMAND_POOL_CREATE_TRANSIENT_BIT indicates 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.

See Also

VkCommandPoolCreateFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandPoolCreateFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCommandPoolResetFlagBits(3)

Name

VkCommandPoolResetFlagBits - Bitmask controlling behavior of a command pool reset

C Specification

Bits which **can** be set in vkResetCommandPool::`flags` to control the reset operation are:

```
typedef enum VkCommandPoolResetFlagBits {
    VK_COMMAND_POOL_RESET_RELEASE_RESOURCES_BIT = 0x00000001,
} VkCommandPoolResetFlagBits;
```

Description

- `VK_COMMAND_POOL_RESET_RELEASE_RESOURCES_BIT` specifies that resetting a command pool recycles all of the resources from the command pool back to the system.

See Also

VkCommandPoolResetFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandPoolResetFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCompareOp(3)

Name

VkCompareOp - Stencil comparison function

C Specification

Possible values of VkStencilOpState::compareOp, specifying the stencil comparison function, are:

```
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;
```

Description

- VK_COMPARE_OP_NEVER specifies that the test never passes.
- VK_COMPARE_OP_LESS specifies that the test passes when R < S.
- VK_COMPARE_OP_EQUAL specifies that the test passes when R = S.
- VK_COMPARE_OP_LESS_OR_EQUAL specifies that the test passes when R ≤ S.
- VK_COMPARE_OP_GREATER specifies that the test passes when R > S.
- VK_COMPARE_OP_NOT_EQUAL specifies that the test passes when R ≠ S.
- VK_COMPARE_OP_GREATER_OR_EQUAL specifies that the test passes when R ≥ S.
- VK_COMPARE_OP_ALWAYS specifies that the test always passes.

See Also

VkPipelineDepthStencilStateCreateInfo, VkSamplerCreateInfo, VkStencilOpState

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCompareOp

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkComponentSwizzle(3)

Name

VkComponentSwizzle - Specify how a component is swizzled

C Specification

Possible values of the members of VkComponentMapping, specifying the component values placed in each component of the output vector, are:

```
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;
```

Description

- `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 10. Component Mappings Equivalent To `VK_COMPONENT_SWIZZLE_IDENTITY`

Component	Identity Mapping
`components.r`	`VK_COMPONENT_SWIZZLE_R`
`components.g`	`VK_COMPONENT_SWIZZLE_G`

Component	Identity Mapping
components.b	VK_COMPONENT_SWIZZLE_B
components.a	VK_COMPONENT_SWIZZLE_A

See Also

VkComponentMapping

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkComponentSwizzle

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkCullModeFlagBits(3)

Name

VkCullModeFlagBits - Bitmask controlling triangle culling

C Specification

Once the orientation of triangles is determined, they are culled according to the VkPipelineRasterizationStateCreateInfo::`cullMode` property of the currently active pipeline. Possible values are:

```
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;
```

Description

- `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.

See Also

VkCullModeFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCullModeFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDependencyFlagBits(3)

Name

VkDependencyFlagBits - Bitmask specifying how execution and memory dependencies are formed

C Specification

Bits which **can** be set in vkCmdPipelineBarrier::dependencyFlags, specifying how execution and memory dependencies are formed, are:

```
typedef enum VkDependencyFlagBits {
    VK_DEPENDENCY_BY_REGION_BIT = 0x00000001,
} VkDependencyFlagBits;
```

Description

- VK_DEPENDENCY_BY_REGION_BIT specifies that dependencies will be framebuffer-local.

See Also

VkDependencyFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDependencyFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorPoolCreateFlagBits(3)

Name

VkDescriptorPoolCreateFlagBits - Bitmask specifying certain supported operations on a descriptor pool

C Specification

Bits which **can** be set in VkDescriptorPoolCreateInfo::`flags` to enable operations on a descriptor pool are:

```
typedef enum VkDescriptorPoolCreateFlagBits {
    VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT = 0x00000001,
} VkDescriptorPoolCreateFlagBits;
```

Description

- `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.

See Also

VkDescriptorPoolCreateFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorPoolCreateFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorSetLayoutCreateFlagBits(3)

Name

VkDescriptorSetLayoutCreateFlagBits - Bitmask specifying descriptor set layout properties

C Specification

Bits which **can** be set in VkDescriptorSetLayoutCreateInfo::`flags` to specify options for descriptor set layout are:

```
typedef enum VkDescriptorSetLayoutCreateFlagBits {
} VkDescriptorSetLayoutCreateFlagBits;
```

Description

See Also

VkDescriptorSetLayoutCreateFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorSetLayoutCreateFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDescriptorType(3)

Name

VkDescriptorType - Specifies the type of a descriptor in a descriptor set

C Specification

The type of descriptors in a descriptor set is specified by VkWriteDescriptorSet::descriptorType, which **must** be one of the values:

```
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;
```

Description

- VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC specify that the elements of the VkWriteDescriptorSet::pBufferInfo array of VkDescriptorBufferInfo structures will be used to update the descriptors, and other arrays will be ignored.
- VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER specify that the VkWriteDescriptorSet::pTexelBufferView array will be used to update the descriptors, and other arrays will be ignored.
- 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 specify that the elements of the VkWriteDescriptorSet ::pImageInfo array of VkDescriptorImageInfo structures will be used to update the descriptors, and other arrays will be ignored.

See Also

VkDescriptorPoolSize, VkDescriptorSetLayoutBinding, VkWriteDescriptorSet

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorType

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDynamicState(3)

Name

VkDynamicState - Indicate which dynamic state is taken from dynamic state commands

C Specification

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:

```
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,
} VkDynamicState;
```

Description

- VK_DYNAMIC_STATE_VIEWPORT specifies that the pViewports state in VkPipelineViewportStateCreateInfo will be ignored and **must** be set dynamically with vkCmdSetViewport before any draw 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 draw 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 draw 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 depthBiasEnable in VkPipelineRasterizationStateCreateInfo set to VK_TRUE.
- 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

See Also

VkPipelineDynamicStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDynamicState

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkFenceCreateFlagBits(3)

Name

VkFenceCreateFlagBits - Bitmask specifying initial state and behavior of a fence

C Specification

```
typedef enum VkFenceCreateFlagBits {
    VK_FENCE_CREATE_SIGNALED_BIT = 0x00000001,
} VkFenceCreateFlagBits;
```

Description

- `VK_FENCE_CREATE_SIGNALED_BIT` specifies that the fence object is created in the signaled state. Otherwise, it is created in the unsignaled state.

See Also

VkFenceCreateFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFenceCreateFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkFilter(3)

Name

VkFilter - Specify filters used for texture lookups

C Specification

Possible values of the VkSamplerCreateInfo::`magFilter` and `minFilter` parameters, specifying filters used for texture lookups, are:

```
typedef enum VkFilter {
    VK_FILTER_NEAREST = 0,
    VK_FILTER_LINEAR = 1,
} VkFilter;
```

Description

- `VK_FILTER_NEAREST` specifies nearest filtering.
- `VK_FILTER_LINEAR` specifies linear filtering.

These filters are described in detail in Texel Filtering.

See Also

VkSamplerCreateInfo, vkCmdBlitImage

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFilter

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkFormat(3)

Name

VkFormat - Available image formats

C Specification

Image formats which **can** be passed to, and **may** be returned from Vulkan commands, are:

```
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,
} VkFormat;
```

Description

- `VK_FORMAT_UNDEFINED` indicates 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_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 html/vkspec.html#fundamentals-fp10 and html/vkspec.html#fundamentals-fp11.
- 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 64-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_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_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_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_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_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_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_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_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_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_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_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_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_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.

See Also

VkAttachmentDescription, VkBufferViewCreateInfo, VkImageCreateInfo, VkImageViewCreateInfo, VkVertexInputAttributeDescription, vkGetPhysicalDeviceFormatProperties, vkGetPhysicalDeviceImageFormatProperties, vkGetPhysicalDeviceSparseImageFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFormat

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkFormatFeatureFlagBits(3)

Name

VkFormatFeatureFlagBits - Bitmask specifying features supported by a buffer

C Specification

Bits which **can** be set in the VkFormatProperties features `linearTilingFeatures`, `optimalTilingFeatures`, and `bufferFeatures` are:

```
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,
} VkFormatFeatureFlagBits;
```

Description

The following bits **may** be set in `linearTilingFeatures` and `optimalTilingFeatures`, specifying that the features are supported by images or image views 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 images.
- `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 and as an input attachment.
- `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

vkCmdBlitImage command.

- VK_FORMAT_FEATURE_BLIT_DST_BIT specifies that an image **can** be used as dstImage for the vkCmdBlitImage command.
- 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 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 indicates 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. If this bit is not present, linear filtering with depth compare disabled is unsupported and linear filtering with depth compare enabled is supported, but **may** compute the filtered value 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.

The following bits **may** be set in bufferFeatures, specifying that the features are supported by buffers or buffer views created with the queried vkGetPhysicalDeviceProperties::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).

See Also

VkFormatFeatureFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFormatFeatureFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkFrontFace(3)

Name

VkFrontFace - Interpret polygon front-facing orientation

C Specification

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:

$$a = -\frac{1}{2}\sum_{i=0}^{n-1} x_f^i y_f^{i \oplus 1} - x_f^{i \oplus 1} y_f^i$$

where x_f^i and y_f^i are the x and y framebuffer coordinates of the ith vertex of the n-vertex polygon (vertices are numbered starting at zero for the purposes of this computation) and i ⊕ 1 is (i + 1) mod n.

The interpretation of the sign of a is determined by the VkPipelineRasterizationStateCreateInfo ::`frontFace` property of the currently active pipeline. Possible values are:

```
typedef enum VkFrontFace {
    VK_FRONT_FACE_COUNTER_CLOCKWISE = 0,
    VK_FRONT_FACE_CLOCKWISE = 1,
} VkFrontFace;
```

Description

- `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.

See Also

VkPipelineRasterizationStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFrontFace

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageAspectFlagBits(3)

Name

VkImageAspectFlagBits - Bitmask specifying which aspects of an image are included in a view

C Specification

Bits which **can** be set in an aspect mask to specify aspects of an image for purposes such as identifying a subresource, are:

```
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,
} VkImageAspectFlagBits;
```

Description

- `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 sparse resource operations.

See Also

VkImageAspectFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageAspectFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageCreateFlagBits(3)

Name

VkImageCreateFlagBits - Bitmask specifying additional parameters of an image

C Specification

Bits which **can** be set in VkImageCreateInfo::`flags`, specifying additional parameters of an image, are:

```
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,
} VkImageCreateFlagBits;
```

Description

- `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` specifies that the image will be backed using sparse memory binding.
- `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.
- `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
- `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` specifies that the image **can** be used to create a `VkImageView` with a different format from 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`.

If any of the bits `VK_IMAGE_CREATE_SPARSE_BINDING_BIT`, `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`, or `VK_IMAGE_CREATE_SPARSE_ALIASED_BIT` are set, `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT` **must** not also be set.

See Sparse Resource Features and Sparse Physical Device Features for more details.

See Also

VkImageCreateFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageCreateFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageLayout(3)

Name

VkImageLayout - Layout of image and image subresources

C Specification

The set of image layouts consists of:

```
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,
} VkImageLayout;
```

Description

The type(s) of device access supported by each layout are:

- `VK_IMAGE_LAYOUT_UNDEFINED` does not support device access. This layout **must** only be used as the `initialLayout` member of `VkImageCreateInfo` or `VkAttachmentDescription`, or as the `oldLayout` in an image transition. When transitioning out of this layout, the contents of the memory are not guaranteed to be preserved.
- `VK_IMAGE_LAYOUT_PREINITIALIZED` does not support device access. This layout **must** only be used as the `initialLayout` member of `VkImageCreateInfo` or `VkAttachmentDescription`, or as the `oldLayout` in an image transition. When transitioning out of this layout, the contents of the memory are preserved. 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 `VK_IMAGE_TILING_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.
- `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` **must** only be used as a depth/stencil attachment in a `VkFramebuffer`. This layout is valid only for image subresources of images created with the `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT` usage bit enabled.

- VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL **must** only be used as a read-only depth/stencil attachment in a VkFramebuffer and/or as a read-only image in a shader (which **can** be read as a sampled image, combined image/sampler and/or input attachment). This layout is valid only for image subresources of images created with the VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT usage bit enabled. Only image subresources of images created with VK_IMAGE_USAGE_SAMPLED_BIT **can** be used as sampled image or combined image/sampler in a shader. Similarly, only image subresources of images created with VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT **can** be used as input attachments.
- VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL **must** only be used as a read-only image in a shader (which **can** be read as a sampled image, combined image/sampler and/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 bit 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.

For each mechanism of accessing an image in the API, there is a parameter or structure member that controls the image layout used to access the image. For transfer commands, this is a parameter to the command (see html/vkspec.html#clears and html/vkspec.html#copies). 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 html/vkspec.html#descriptorsets-updates). 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 the layout specified via the API controlling those accesses.

The image layout of each image subresource **must** be well-defined at each point in the image subresource's lifetime. This means that when performing a layout transition on the 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 VK_IMAGE_LAYOUT_UNDEFINED or VK_IMAGE_LAYOUT_PREINITIALIZED.

See Also

VkAttachmentDescription, VkAttachmentReference, VkDescriptorImageInfo, VkImageCreateInfo, VkImageMemoryBarrier, vkCmdBlitImage, vkCmdClearColorImage, vkCmdClearDepthStencilImage, vkCmdCopyBufferToImage, vkCmdCopyImage, vkCmdCopyImageToBuffer, vkCmdResolveImage

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageLayout

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageTiling(3)

Name

VkImageTiling - Specifies the tiling arrangement of data in an image

C Specification

Possible values of VkImageCreateInfo::tiling, specifying the tiling arrangement of data elements in an image, are:

```
typedef enum VkImageTiling {
    VK_IMAGE_TILING_OPTIMAL = 0,
    VK_IMAGE_TILING_LINEAR = 1,
} VkImageTiling;
```

Description

- VK_IMAGE_TILING_OPTIMAL specifies optimal tiling (texels are laid out in an implementation-dependent arrangement, for more optimal 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).

See Also

VkImageCreateInfo, vkGetPhysicalDeviceImageFormatProperties, vkGetPhysicalDeviceSparseImageFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageTiling

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageType(3)

Name

VkImageType - Specifies the type of an image object

C Specification

Possible values of VkImageCreateInfo::imageType, specifying the basic dimensionality of an image, are:

```
typedef enum VkImageType {
    VK_IMAGE_TYPE_1D = 0,
    VK_IMAGE_TYPE_2D = 1,
    VK_IMAGE_TYPE_3D = 2,
} VkImageType;
```

Description

- 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.

See Also

VkImageCreateInfo, vkGetPhysicalDeviceImageFormatProperties, vkGetPhysicalDeviceSparseImageFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageType

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageUsageFlagBits(3)

Name

VkImageUsageFlagBits - Bitmask specifying intended usage of an image

C Specification

Bits which **can** be set in VkImageCreateInfo::usage, specifying intended usage of an image, are:

```
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,
} VkImageUsageFlagBits;
```

Description

- 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 attachment in a VkFramebuffer.
- VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT specifies that the memory bound to this image will have been allocated with the VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT (see html/vkspec.html#memory for more detail). 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.

See Also

VkImageUsageFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageUsageFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkImageViewType(3)

Name

VkImageViewType - Image view types

C Specification

The types of image views that **can** be created are:

```
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;
```

Description

The exact image view type is partially implicit, based on the image's type and sample count, as well as the view creation parameters as described in the image view compatibility table for vkCreateImageView. This table also shows which SPIR-V `OpTypeImage` `Dim` and `Arrayed` parameters correspond to each image view type.

See Also

VkImageViewCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageViewType

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkIndexType(3)

Name

VkIndexType - Type of index buffer indices

C Specification

Possible values of vkCmdBindIndexBuffer::`indexType`, specifying the size of indices, are:

```
typedef enum VkIndexType {
    VK_INDEX_TYPE_UINT16 = 0,
    VK_INDEX_TYPE_UINT32 = 1,
} VkIndexType;
```

Description

- `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.

See Also

vkCmdBindIndexBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkIndexType

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkInternalAllocationType(3)

Name

VkInternalAllocationType - Allocation type

C Specification

The `allocationType` parameter to the `pfnInternalAllocation` and `pfnInternalFree` functions **may** be one of the following values:

```
typedef enum VkInternalAllocationType {
    VK_INTERNAL_ALLOCATION_TYPE_EXECUTABLE = 0,
} VkInternalAllocationType;
```

Description

- `VK_INTERNAL_ALLOCATION_TYPE_EXECUTABLE` specifies that the allocation is intended for execution by the host.

See Also

PFN_vkInternalAllocationNotification, PFN_vkInternalFreeNotification

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkInternalAllocationType

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkLogicOp(3)

Name

VkLogicOp - Framebuffer logical operations

C Specification

Logical operations are controlled by the `logicOpEnable` and `logicOp` members of VkPipelineColorBlendStateCreateInfo. 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:

```
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;
```

Description

The logical operations supported by Vulkan are summarized in the following table in which

- ¬ is bitwise invert,
- ∧ is bitwise and,
- ∨ is bitwise or,
- ⊕ 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 11. Logical Operations

Mode	Operation
VK_LOGIC_OP_CLEAR	0
VK_LOGIC_OP_AND	s ∧ d
VK_LOGIC_OP_AND_REVERSE	s ∧ ¬ d
VK_LOGIC_OP_COPY	s
VK_LOGIC_OP_AND_INVERTED	¬ s ∧ d
VK_LOGIC_OP_NO_OP	d
VK_LOGIC_OP_XOR	s ⊕ d
VK_LOGIC_OP_OR	s ∨ d
VK_LOGIC_OP_NOR	¬ (s ∨ d)
VK_LOGIC_OP_EQUIVALENT	¬ (s ⊕ d)
VK_LOGIC_OP_INVERT	¬ d
VK_LOGIC_OP_OR_REVERSE	s ∨ ¬ d
VK_LOGIC_OP_COPY_INVERTED	¬ s
VK_LOGIC_OP_OR_INVERTED	¬ s ∨ d
VK_LOGIC_OP_NAND	¬ (s ∧ 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.

See Also

VkPipelineColorBlendStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkLogicOp

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkMemoryHeapFlagBits(3)

Name

VkMemoryHeapFlagBits - Bitmask specifying attribute flags for a heap

C Specification

Bits which **may** be set in VkMemoryHeap::`flags`, indicating attribute flags for the heap, are:

```
typedef enum VkMemoryHeapFlagBits {
    VK_MEMORY_HEAP_DEVICE_LOCAL_BIT = 0x00000001,
} VkMemoryHeapFlagBits;
```

Description

- `VK_MEMORY_HEAP_DEVICE_LOCAL_BIT` indicates 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.

See Also

VkMemoryHeapFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkMemoryHeapFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkMemoryPropertyFlagBits(3)

Name

VkMemoryPropertyFlagBits - Bitmask specifying properties for a memory type

C Specification

Bits which **may** be set in VkMemoryType::propertyFlags, indicating properties of a memory heap, are:

```
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,
} VkMemoryPropertyFlagBits;
```

Description

- `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT` bit indicates that memory allocated with this type is the most efficient for device access. This property will only be set for memory types belonging to heaps with the `VK_MEMORY_HEAP_DEVICE_LOCAL_BIT` set.
- `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT` bit indicates that memory allocated with this type **can** be mapped for host access using vkMapMemory.
- `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT` bit indicates that the host cache management commands vkFlushMappedMemoryRanges and vkInvalidateMappedMemoryRanges are not needed to flush host writes to the device or make device writes visible to the host, respectively.
- `VK_MEMORY_PROPERTY_HOST_CACHED_BIT` bit indicates 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 indicates 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.

See Also

VkMemoryPropertyFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkMemoryPropertyFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkObjectType(3)

Name

VkObjectType - Specify an enumeration to track object handle types

C Specification

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.

```
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,
} VkObjectType;
```

Description

See Also

No cross-references are available

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkObjectType

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPhysicalDeviceType(3)

Name

VkPhysicalDeviceType - Supported physical device types

C Specification

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

```
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;
```

Description

- `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.

See Also

VkPhysicalDeviceProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPhysicalDeviceType

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineBindPoint(3)

Name

VkPipelineBindPoint - Specify the bind point of a pipeline object to a command buffer

C Specification

Possible values of vkCmdBindPipeline::`pipelineBindPoint`, specifying the bind point of a pipeline object, are:

```
typedef enum VkPipelineBindPoint {
    VK_PIPELINE_BIND_POINT_GRAPHICS = 0,
    VK_PIPELINE_BIND_POINT_COMPUTE = 1,
} VkPipelineBindPoint;
```

Description

- `VK_PIPELINE_BIND_POINT_COMPUTE` specifies binding as a compute pipeline.
- `VK_PIPELINE_BIND_POINT_GRAPHICS` specifies binding as a graphics pipeline.

See Also

VkSubpassDescription, vkCmdBindDescriptorSets, vkCmdBindPipeline

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineBindPoint

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineCacheHeaderVersion(3)

Name

VkPipelineCacheHeaderVersion - Encode pipeline cache version

C Specification

Possible values of the second group of four bytes in the header returned by vkGetPipelineCacheData, encoding the pipeline cache version, are:

```
typedef enum VkPipelineCacheHeaderVersion {
    VK_PIPELINE_CACHE_HEADER_VERSION_ONE = 1,
} VkPipelineCacheHeaderVersion;
```

Description

- `VK_PIPELINE_CACHE_HEADER_VERSION_ONE` specifies version one of the pipeline cache.

See Also

vkCreatePipelineCache, vkGetPipelineCacheData

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineCacheHeaderVersion

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineCreateFlagBits(3)

Name

VkPipelineCreateFlagBits - Bitmask controlling how a pipeline is created

C Specification

Possible values of the `flags` member of VkGraphicsPipelineCreateInfo and VkComputePipelineCreateInfo, specifying how a pipeline is created, are:

```
typedef enum VkPipelineCreateFlagBits {
    VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT = 0x00000001,
    VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT = 0x00000002,
    VK_PIPELINE_CREATE_DERIVATIVE_BIT = 0x00000004,
} VkPipelineCreateFlagBits;
```

Description

- `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_ALLOW_DERIVATIVES_BIT` specifies that the pipeline to be created is allowed to be the parent of a pipeline that will be created in a subsequent call to vkCreateGraphicsPipelines.
- `VK_PIPELINE_CREATE_DERIVATIVE_BIT` specifies that the pipeline to be created will be a child of a previously created parent pipeline.

It is valid to set both `VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT` and `VK_PIPELINE_CREATE_DERIVATIVE_BIT`. This allows a pipeline to be both a parent and possibly a child in a pipeline hierarchy. See Pipeline Derivatives for more information.

See Also

VkPipelineCreateFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineCreateFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPipelineStageFlagBits(3)

Name

VkPipelineStageFlagBits - Bitmask specifying pipeline stages

C Specification

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, specifying pipeline stages, are:

```
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,
} VkPipelineStageFlagBits;
```

Description

- `VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT` specifies the stage of the pipeline where any commands are initially received by the queue.
- `VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT` specifies the stage of the pipeline where Draw/DispatchIndirect 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 subpass 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 subpass 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 also includes subpass load and store operations and multisample resolve operations for framebuffer attachments with a color format.
- VK_PIPELINE_STAGE_TRANSFER_BIT specifies the execution of copy commands. This includes the operations resulting from all copy commands, clear commands (with the exception of vkCmdClearAttachments), and vkCmdCopyQueryPoolResults.
- VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT specifies the execution of a compute shader.
- VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT specifies the final stage in the pipeline where operations generated by all commands complete execution.
- 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_TOP_OF_PIPE_BIT
  - 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_BOTTOM_OF_PIPE_BIT
- VK_PIPELINE_STAGE_ALL_COMMANDS_BIT is equivalent to the logical OR of every other pipeline stage flag that is supported on the queue it is used with.

> *Note*
>
> An execution dependency with only `VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT` in the destination stage mask will only prevent that stage from executing in subsequently submitted commands. As this stage does not perform any actual execution, this is not observable - in effect, it does not delay processing of subsequent commands. Similarly an execution dependency with only `VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT` in the source stage mask will effectively not wait for any prior commands to complete.
>
> When defining a memory dependency, using only `VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT` or `VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT` would never make any accesses available and/or visible because these stages do not access memory.
>
> `VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT` and `VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT` are useful for accomplishing layout transitions and queue ownership operations when the required execution dependency is satisfied by other means - for example, semaphore operations between queues.

See Also

VkPipelineStageFlags, vkCmdWriteTimestamp

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineStageFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPolygonMode(3)

Name

VkPolygonMode - Control polygon rasterization mode

C Specification

Possible values of the VkPipelineRasterizationStateCreateInfo::polygonMode property of the currently active pipeline, specifying the method of rasterization for polygons, are:

```
typedef enum VkPolygonMode {
    VK_POLYGON_MODE_FILL = 0,
    VK_POLYGON_MODE_LINE = 1,
    VK_POLYGON_MODE_POINT = 2,
} VkPolygonMode;
```

Description

- 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.

See Also

VkPipelineRasterizationStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPolygonMode

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkPrimitiveTopology(3)

Name

VkPrimitiveTopology - Supported primitive topologies

C Specification

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. Supported topologies are defined by VkPrimitiveTopology and include:

```
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;
```

Description

See Also

VkPipelineInputAssemblyStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPrimitiveTopology

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkQueryControlFlagBits(3)

Name

VkQueryControlFlagBits - Bitmask specifying constraints on a query

C Specification

Bits which **can** be set in vkCmdBeginQuery::flags, specifying constraints on the types of queries that **can** be performed, are:

```
typedef enum VkQueryControlFlagBits {
    VK_QUERY_CONTROL_PRECISE_BIT = 0x00000001,
} VkQueryControlFlagBits;
```

Description

- VK_QUERY_CONTROL_PRECISE_BIT specifies the precision of occlusion queries.

See Also

VkQueryControlFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueryControlFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkQueryPipelineStatisticFlagBits(3)

Name

VkQueryPipelineStatisticFlagBits - Bitmask specifying queried pipeline statistics

C Specification

Bits which **can** be set to individually enable pipeline statistics counters for query pools with VkQueryPoolCreateInfo::pipelineStatistics, and for secondary command buffers with VkCommandBufferInheritanceInfo::pipelineStatistics, are:

```
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;
```

Description

- 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 vertex processing 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`).

See Also

VkQueryPipelineStatisticFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueryPipelineStatisticFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkQueryResultFlagBits(3)

Name

VkQueryResultFlagBits - Bitmask specifying how and when query results are returned

C Specification

Bits which **can** be set in vkGetQueryPoolResults::`flags` and vkCmdCopyQueryPoolResults::`flags`, specifying how and when results are returned, are:

```
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;
```

Description

- `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.

See Also

VkQueryResultFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueryResultFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkQueryType(3)

Name

VkQueryType - Specify the type of queries managed by a query pool

C Specification

Possible values of VkQueryPoolCreateInfo::`queryType`, specifying the type of queries managed by the pool, are:

```
typedef enum VkQueryType {
    VK_QUERY_TYPE_OCCLUSION = 0,
    VK_QUERY_TYPE_PIPELINE_STATISTICS = 1,
    VK_QUERY_TYPE_TIMESTAMP = 2,
} VkQueryType;
```

Description

- `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.

See Also

VkQueryPoolCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueryType

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkQueueFlagBits(3)

Name

VkQueueFlagBits - Bitmask specifying capabilities of queues in a queue family

C Specification

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

```
typedef enum VkQueueFlagBits {
    VK_QUEUE_GRAPHICS_BIT = 0x00000001,
    VK_QUEUE_COMPUTE_BIT = 0x00000002,
    VK_QUEUE_TRANSFER_BIT = 0x00000004,
    VK_QUEUE_SPARSE_BINDING_BIT = 0x00000008,
} VkQueueFlagBits;
```

Description

- VK_QUEUE_GRAPHICS_BIT indicates that queues in this queue family support graphics operations.
- VK_QUEUE_COMPUTE_BIT indicates that queues in this queue family support compute operations.
- VK_QUEUE_TRANSFER_BIT indicates that queues in this queue family support transfer operations.
- VK_QUEUE_SPARSE_BINDING_BIT indicates 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.

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.

> *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.

See Also

VkQueueFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueueFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkResult(3)

Name

VkResult - Vulkan command return codes

C Specification

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.
- Run time error codes are returned when a command needs to communicate a failure that could only be detected at run time. All run time error codes are negative values.

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

```
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,
} VkResult;
```

Description

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

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 accomodate the new allocation.

If a command returns a run time error, it will leave any result pointers unmodified, unless other behavior is explicitly defined in the specification.

Out of memory errors do not damage any currently existing Vulkan objects. Objects that have already been successfully created **can** still be used by the application.

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

See Also

No cross-references are available

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkResult

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the

Specification, not directly.

VkSampleCountFlagBits(3)

Name

VkSampleCountFlagBits - Bitmask specifying sample counts supported for an image used for storage operations

C Specification

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:

```
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;
```

Description

- `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.

See Also

VkAttachmentDescription, VkImageCreateInfo, VkPipelineMultisampleStateCreateInfo, VkSampleCountFlags, vkGetPhysicalDeviceSparseImageFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSampleCountFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSamplerAddressMode(3)

Name

VkSamplerAddressMode - Specify behavior of sampling with texture coordinates outside an image

C Specification

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:

```
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,
    VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE = 4,
} VkSamplerAddressMode;
```

Description

- `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 VK_KHR_mirror_clamp_to_edge extension is enabled.

See Also

VkSamplerCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSamplerAddressMode

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSamplerMipmapMode(3)

Name

VkSamplerMipmapMode - Specify mipmap mode used for texture lookups

C Specification

Possible values of the VkSamplerCreateInfo::mipmapMode, specifying the mipmap mode used for texture lookups, are:

```
typedef enum VkSamplerMipmapMode {
    VK_SAMPLER_MIPMAP_MODE_NEAREST = 0,
    VK_SAMPLER_MIPMAP_MODE_LINEAR = 1,
} VkSamplerMipmapMode;
```

Description

- `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.

See Also

VkSamplerCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSamplerMipmapMode

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkShaderStageFlagBits(3)

Name

VkShaderStageFlagBits - Bitmask specifying a pipeline stage

C Specification

Commands and structures which need to specify one or more shader stages do so using a bitmask whose bits correspond to stages. Bits which **can** be set to specify shader stages are:

```
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;
```

Description

- `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.

See Also

VkPipelineShaderStageCreateInfo, VkShaderStageFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkShaderStageFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSharingMode(3)

Name

VkSharingMode - Buffer and image sharing modes

C Specification

Buffer and image objects are created with a *sharing mode* controlling how they **can** be accessed from queues. The supported sharing modes are:

```
typedef enum VkSharingMode {
    VK_SHARING_MODE_EXCLUSIVE = 0,
    VK_SHARING_MODE_CONCURRENT = 1,
} VkSharingMode;
```

Description

- `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 same queue family at any given time. In order for a different queue family to be able to interpret the memory contents of a range or image subresource, the application **must** perform a queue family ownership transfer.

Upon creation, resources using `VK_SHARING_MODE_EXCLUSIVE` are not owned by any queue family. A buffer or image memory barrier is not required to acquire *ownership* when no queue family owns the resource - it is implicitly acquired upon first use within a queue.

> *Note*
>
> Images still require a layout transition from `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED` before being used on the first queue.

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.

See Also

VkBufferCreateInfo, VkImageCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSharingMode

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSparseImageFormatFlagBits(3)

Name

VkSparseImageFormatFlagBits - Bitmask specifying additional information about a sparse image resource

C Specification

Bits which **can** be set in VkSparseImageFormatProperties::flags, specifying additional information about the sparse resource, are:

```
typedef enum VkSparseImageFormatFlagBits {
    VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT = 0x00000001,
    VK_SPARSE_IMAGE_FORMAT_ALIGNED_MIP_SIZE_BIT = 0x00000002,
    VK_SPARSE_IMAGE_FORMAT_NONSTANDARD_BLOCK_SIZE_BIT = 0x00000004,
} VkSparseImageFormatFlagBits;
```

Description

- VK_SPARSE_IMAGE_FORMAT_SINGLE_MIPTAIL_BIT specifies that the image uses a single mip tail region for all array layers.
- VK_SPARSE_IMAGE_FORMAT_ALIGNED_MIP_SIZE_BIT specifies that the first mip level whose dimensions are not integer multiples of the corresponding dimensions of the sparse image block begins the mip tail region.
- VK_SPARSE_IMAGE_FORMAT_NONSTANDARD_BLOCK_SIZE_BIT specifies that the image uses non-standard sparse image block dimensions, and the imageGranularity values do not match the standard sparse image block dimensions for the given pixel format.

See Also

VkSparseImageFormatFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSparseImageFormatFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSparseMemoryBindFlagBits(3)

Name

VkSparseMemoryBindFlagBits - Bitmask specifying usage of a sparse memory binding operation

C Specification

Bits which **can** be set in VkSparseMemoryBind::`flags`, specifying usage of a sparse memory binding operation, are:

```
typedef enum VkSparseMemoryBindFlagBits {
    VK_SPARSE_MEMORY_BIND_METADATA_BIT = 0x00000001,
} VkSparseMemoryBindFlagBits;
```

Description

- `VK_SPARSE_MEMORY_BIND_METADATA_BIT` specifies that the memory being bound is only for the metadata aspect.

See Also

VkSparseMemoryBindFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSparseMemoryBindFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkStencilFaceFlagBits(3)

Name

VkStencilFaceFlagBits - Bitmask specifying sets of stencil state for which to update the compare mask

C Specification

Bits which **can** be set in the vkCmdSetStencilCompareMask::`faceMask` parameter, and similar parameters of other commands specifying which stencil state to update stencil masks for, are:

```
typedef enum VkStencilFaceFlagBits {
    VK_STENCIL_FACE_FRONT_BIT = 0x00000001,
    VK_STENCIL_FACE_BACK_BIT = 0x00000002,
    VK_STENCIL_FRONT_AND_BACK = 0x00000003,
} VkStencilFaceFlagBits;
```

Description

- `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_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.

See Also

VkStencilFaceFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkStencilFaceFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkStencilOp(3)

Name

VkStencilOp - Stencil comparison function

C Specification

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:

```
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;
```

Description

- `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.

If the stencil test fails, the sample's coverage bit is cleared in the fragment. If there is no stencil framebuffer attachment, stencil modification **cannot** occur, and it is as if the stencil tests always pass.

If the stencil test passes, the `writeMask` member of the VkStencilOpState structures controls how the updated stencil value is written to the stencil framebuffer attachment.

The least significant s bits of `writeMask`, where s is the number of bits in the stencil framebuffer attachment, specify an integer mask. Where a 1 appears in this mask, the corresponding bit in the stencil value in the depth/stencil attachment is written; where a 0 appears, the bit is not written. The `writeMask` value uses either the front-facing or back-facing state based on the facingness of the fragment. Fragments generated by front-facing primitives use the front mask and fragments generated by back-facing primitives use the back mask.

See Also

VkStencilOpState

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkStencilOp

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkStructureType(3)

Name

VkStructureType - Vulkan structure types (`stype`)

C Specification

Vulkan structures containing `sType` members **must** have a value of `sType` matching the type of the structure, as described more fully in Valid Usage for Structure Types. Structure types supported by the Vulkan API include:

```
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_BIND_SPARSE_INFO = 7,
    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_SHADER_MODULE_CREATE_INFO = 16,
    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,
} VkStructureType;
```

Description

See Also

VkApplicationInfo, VkBindSparseInfo, VkBufferCreateInfo, VkBufferMemoryBarrier, VkBufferViewCreateInfo, VkCommandBufferAllocateInfo, VkCommandBufferBeginInfo, VkCommandBufferInheritanceInfo, VkCommandPoolCreateInfo, VkComputePipelineCreateInfo, VkCopyDescriptorSet, VkDescriptorPoolCreateInfo, VkDescriptorSetAllocateInfo, VkDescriptorSetLayoutCreateInfo, VkDeviceCreateInfo, VkDeviceQueueCreateInfo, VkEventCreateInfo, VkFenceCreateInfo, VkFramebufferCreateInfo, VkGraphicsPipelineCreateInfo, VkImageCreateInfo, VkImageMemoryBarrier, VkImageViewCreateInfo, VkInstanceCreateInfo, VkMappedMemoryRange, VkMemoryAllocateInfo, VkMemoryBarrier, VkPipelineCacheCreateInfo, VkPipelineColorBlendStateCreateInfo, VkPipelineDepthStencilStateCreateInfo, VkPipelineDynamicStateCreateInfo, VkPipelineInputAssemblyStateCreateInfo, VkPipelineLayoutCreateInfo, VkPipelineMultisampleStateCreateInfo, VkPipelineRasterizationStateCreateInfo, VkPipelineShaderStageCreateInfo, VkPipelineTessellationStateCreateInfo, VkPipelineVertexInputStateCreateInfo, VkPipelineViewportStateCreateInfo, VkQueryPoolCreateInfo, VkRenderPassBeginInfo, VkRenderPassCreateInfo, VkSamplerCreateInfo, VkSemaphoreCreateInfo, VkShaderModuleCreateInfo, VkSubmitInfo, VkWriteDescriptorSet

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkStructureType

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSubpassContents(3)

Name

VkSubpassContents - Specify how commands in the first subpass of a render pass are provided

C Specification

Possible values of vkCmdBeginRenderPass::`contents`, specifying how the commands in the first subpass will be provided, are:

```
typedef enum VkSubpassContents {
    VK_SUBPASS_CONTENTS_INLINE = 0,
    VK_SUBPASS_CONTENTS_SECONDARY_COMMAND_BUFFERS = 1,
} VkSubpassContents;
```

Description

- `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 on the command buffer until vkCmdNextSubpass or vkCmdEndRenderPass.

See Also

vkCmdBeginRenderPass, vkCmdNextSubpass

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSubpassContents

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSubpassDescriptionFlagBits(3)

Name

VkSubpassDescriptionFlagBits - Bitmask specifying usage of a subpass

C Specification

Bits which **can** be set in VkSubpassDescription::`flags`, specifying usage of the subpass, are:

```
typedef enum VkSubpassDescriptionFlagBits {
} VkSubpassDescriptionFlagBits;
```

Description

See Also

VkSubpassDescriptionFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSubpassDescriptionFlagBits

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSystemAllocationScope(3)

Name

VkSystemAllocationScope - Allocation scope

C Specification

Each allocation has an *allocation scope* which defines its lifetime and which object it is associated with. Possible values passed to the `allocationScope` parameter of the callback functions specified by VkAllocationCallbacks, indicating the allocation scope, are:

```
typedef enum VkSystemAllocationScope {
    VK_SYSTEM_ALLOCATION_SCOPE_COMMAND = 0,
    VK_SYSTEM_ALLOCATION_SCOPE_OBJECT = 1,
    VK_SYSTEM_ALLOCATION_SCOPE_CACHE = 2,
    VK_SYSTEM_ALLOCATION_SCOPE_DEVICE = 3,
    VK_SYSTEM_ALLOCATION_SCOPE_INSTANCE = 4,
} VkSystemAllocationScope;
```

Description

- `VK_SYSTEM_ALLOCATION_SCOPE_COMMAND` specifies that the allocation is scoped to the duration of the Vulkan command.
- `VK_SYSTEM_ALLOCATION_SCOPE_OBJECT` specifies that the allocation is scoped to the lifetime of the Vulkan object that is being created or used.
- `VK_SYSTEM_ALLOCATION_SCOPE_CACHE` specifies that the allocation is scoped to the lifetime of a `VkPipelineCache` object.
- `VK_SYSTEM_ALLOCATION_SCOPE_DEVICE` specifies that the allocation is scoped to the lifetime of the Vulkan device.
- `VK_SYSTEM_ALLOCATION_SCOPE_INSTANCE` specifies that the allocation is scoped to the lifetime of the Vulkan instance.

Most Vulkan commands operate on a single object, or there is a sole object that is being created or manipulated. When an allocation uses an allocation scope of `VK_SYSTEM_ALLOCATION_SCOPE_OBJECT` or `VK_SYSTEM_ALLOCATION_SCOPE_CACHE`, the allocation is scoped to the object being created or manipulated.

When an implementation requires host memory, it will make callbacks to the application using the most specific allocator and allocation scope available:

- If an allocation is scoped to the duration of a command, the allocator will use the `VK_SYSTEM_ALLOCATION_SCOPE_COMMAND` allocation scope. The most specific allocator available is used: if the object being created or manipulated has an allocator, that object's allocator will be used, else if the parent `VkDevice` has an allocator it will be used, else if the parent `VkInstance` has an allocator it will be used. Else,

- If an allocation is associated with an object of type `VkPipelineCache`, the allocator will use the `VK_SYSTEM_ALLOCATION_SCOPE_CACHE` allocation scope. The most specific allocator available is used (pipeline cache, else device, else instance). Else,
- If an allocation is scoped to the lifetime of an object, that object is being created or manipulated by the command, and that object's type is not `VkDevice` or `VkInstance`, the allocator will use an allocation scope of `VK_SYSTEM_ALLOCATION_SCOPE_OBJECT`. The most specific allocator available is used (object, else device, else instance). Else,
- If an allocation is scoped to the lifetime of a device, the allocator will use an allocation scope of `VK_SYSTEM_ALLOCATION_SCOPE_DEVICE`. The most specific allocator available is used (device, else instance). Else,
- If the allocation is scoped to the lifetime of an instance and the instance has an allocator, its allocator will be used with an allocation scope of `VK_SYSTEM_ALLOCATION_SCOPE_INSTANCE`.
- Otherwise an implementation will allocate memory through an alternative mechanism that is unspecified.

See Also

VkAllocationCallbacks

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSystemAllocationScope

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkVertexInputRate(3)

Name

VkVertexInputRate - Specify rate at which vertex attributes are pulled from buffers

C Specification

Possible values of VkVertexInputBindingDescription::inputRate, specifying the rate at which vertex attributes are pulled from buffers, are:

```
typedef enum VkVertexInputRate {
    VK_VERTEX_INPUT_RATE_VERTEX = 0,
    VK_VERTEX_INPUT_RATE_INSTANCE = 1,
} VkVertexInputRate;
```

Description

- 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.

See Also

VkVertexInputBindingDescription

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkVertexInputRate

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

Flags

VkAccessFlags(3)

Name

VkAccessFlags - Bitmask of VkAccessFlagBits

C Specification

```
typedef VkFlags VkAccessFlags;
```

Description

`VkAccessFlags` is a mask of zero or more VkAccessFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkAccessFlagBits, VkBufferMemoryBarrier, VkImageMemoryBarrier, VkMemoryBarrier, VkSubpassDependency

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkAccessFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkAttachmentDescriptionFlags(3)

Name

VkAttachmentDescriptionFlags - Bitmask of VkAttachmentDescriptionFlagBits

C Specification

```
typedef VkFlags VkAttachmentDescriptionFlags;
```

Description

`VkAttachmentDescriptionFlags` is a mask of zero or more VkAttachmentDescriptionFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkAttachmentDescription, VkAttachmentDescriptionFlagBits

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkAttachmentDescriptionFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkBufferCreateFlags(3)

Name

VkBufferCreateFlags - Bitmask of VkBufferCreateFlagBits

C Specification

```
typedef VkFlags VkBufferCreateFlags;
```

Description

`VkBufferCreateFlags` is a mask of zero or more VkBufferCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkBufferCreateFlagBits, VkBufferCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBufferCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkBufferUsageFlags(3)

Name

VkBufferUsageFlags - Bitmask of VkBufferUsageFlagBits

C Specification

```
typedef VkFlags VkBufferUsageFlags;
```

Description

`VkBufferUsageFlags` is a mask of zero or more VkBufferUsageFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkBufferCreateInfo, VkBufferUsageFlagBits

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBufferUsageFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkBufferViewCreateFlags(3)

Name

VkBufferViewCreateFlags - Bitmask of VkBufferViewCreateFlagBits

C Specification

```
typedef VkFlags VkBufferViewCreateFlags;
```

Description

`VkBufferViewCreateFlags` is a mask of zero or more VkBufferViewCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkBufferViewCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBufferViewCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkColorComponentFlags(3)

Name

VkColorComponentFlags - Bitmask of VkColorComponentFlagBits

C Specification

```
typedef VkFlags VkColorComponentFlags;
```

Description

`VkColorComponentFlags` is a mask of zero or more VkColorComponentFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkColorComponentFlagBits, VkPipelineColorBlendAttachmentState

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkColorComponentFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkCommandBufferResetFlags(3)

Name

VkCommandBufferResetFlags - Bitmask of VkCommandBufferResetFlagBits

C Specification

```
typedef VkFlags VkCommandBufferResetFlags;
```

Description

`VkCommandBufferResetFlags` is a mask of zero or more VkCommandBufferResetFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkCommandBufferResetFlagBits, vkResetCommandBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandBufferResetFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkCommandBufferUsageFlags(3)

Name

VkCommandBufferUsageFlags - Bitmask of VkCommandBufferUsageFlagBits

C Specification

```
typedef VkFlags VkCommandBufferUsageFlags;
```

Description

`VkCommandBufferUsageFlags` is a mask of zero or more VkCommandBufferUsageFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkCommandBufferBeginInfo, VkCommandBufferUsageFlagBits

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandBufferUsageFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkCommandPoolCreateFlags(3)

Name

VkCommandPoolCreateFlags - Bitmask of VkCommandPoolCreateFlagBits

C Specification

```
typedef VkFlags VkCommandPoolCreateFlags;
```

Description

`VkCommandPoolCreateFlags` is a mask of zero or more VkCommandPoolCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkCommandPoolCreateFlagBits, VkCommandPoolCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandPoolCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkCommandPoolResetFlags(3)

Name

VkCommandPoolResetFlags - Bitmask of VkCommandPoolResetFlagBits

C Specification

```
typedef VkFlags VkCommandPoolResetFlags;
```

Description

`VkCommandPoolResetFlags` is a mask of zero or more VkCommandPoolResetFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkCommandPoolResetFlagBits, vkResetCommandPool

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCommandPoolResetFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkCullModeFlags(3)

Name

VkCullModeFlags - Bitmask of VkCullModeFlagBits

C Specification

```
typedef VkFlags VkCullModeFlags;
```

Description

`VkCullModeFlags` is a mask of zero or more VkCullModeFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkCullModeFlagBits, VkPipelineRasterizationStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkCullModeFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkDependencyFlags(3)

Name

VkDependencyFlags - Bitmask of VkDependencyFlagBits

C Specification

```
typedef VkFlags VkDependencyFlags;
```

Description

`VkDependencyFlags` is a mask of zero or more VkDependencyFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkDependencyFlagBits, VkSubpassDependency, vkCmdPipelineBarrier

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDependencyFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkDescriptorPoolCreateFlags(3)

Name

VkDescriptorPoolCreateFlags - Bitmask of VkDescriptorPoolCreateFlagBits

C Specification

```
typedef VkFlags VkDescriptorPoolCreateFlags;
```

Description

`VkDescriptorPoolCreateFlags` is a mask of zero or more VkDescriptorPoolCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkDescriptorPoolCreateFlagBits, VkDescriptorPoolCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorPoolCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkDescriptorPoolResetFlags(3)

Name

VkDescriptorPoolResetFlags - Bitmask of VkDescriptorPoolResetFlagBits

C Specification

```
typedef VkFlags VkDescriptorPoolResetFlags;
```

Description

`VkDescriptorPoolResetFlags` is a mask of zero or more VkDescriptorPoolResetFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

vkResetDescriptorPool

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorPoolResetFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkDescriptorSetLayoutCreateFlags(3)

Name

VkDescriptorSetLayoutCreateFlags - Bitmask of VkDescriptorSetLayoutCreateFlagBits

C Specification

```
typedef VkFlags VkDescriptorSetLayoutCreateFlags;
```

Description

`VkDescriptorSetLayoutCreateFlags` is a mask of zero or more VkDescriptorSetLayoutCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkDescriptorSetLayoutCreateFlagBits, VkDescriptorSetLayoutCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDescriptorSetLayoutCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkDeviceCreateFlags(3)

Name

VkDeviceCreateFlags - Bitmask of VkDeviceCreateFlagBits

C Specification

```
typedef VkFlags VkDeviceCreateFlags;
```

Description

`VkDeviceCreateFlags` is a mask of zero or more VkDeviceCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkDeviceCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDeviceCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkDeviceQueueCreateFlags(3)

Name

VkDeviceQueueCreateFlags - Bitmask of VkDeviceQueueCreateFlagBits

C Specification

```
typedef VkFlags VkDeviceQueueCreateFlags;
```

Description

`VkDeviceQueueCreateFlags` is a mask of zero or more VkDeviceQueueCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkDeviceQueueCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDeviceQueueCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkEventCreateFlags(3)

Name

VkEventCreateFlags - Bitmask of VkEventCreateFlagBits

C Specification

```
typedef VkFlags VkEventCreateFlags;
```

Description

`VkEventCreateFlags` is a mask of zero or more VkEventCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkEventCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkEventCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkFenceCreateFlags(3)

Name

VkFenceCreateFlags - Bitmask of VkFenceCreateFlagBits

C Specification

```
typedef VkFlags VkFenceCreateFlags;
```

Description

`VkFenceCreateFlags` is a mask of zero or more VkFenceCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkFenceCreateFlagBits, VkFenceCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFenceCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkFormatFeatureFlags(3)

Name

VkFormatFeatureFlags - Bitmask of VkFormatFeatureFlagBits

C Specification

```
typedef VkFlags VkFormatFeatureFlags;
```

Description

`VkFormatFeatureFlags` is a mask of zero or more VkFormatFeatureFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkFormatFeatureFlagBits, VkFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFormatFeatureFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkFramebufferCreateFlags(3)

Name

VkFramebufferCreateFlags - Bitmask of VkFramebufferCreateFlagBits

C Specification

```
typedef VkFlags VkFramebufferCreateFlags;
```

Description

`VkFramebufferCreateFlags` is a mask of zero or more VkFramebufferCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkFramebufferCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFramebufferCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkImageAspectFlags(3)

Name

VkImageAspectFlags - Bitmask of VkImageAspectFlagBits

C Specification

```
typedef VkFlags VkImageAspectFlags;
```

Description

`VkImageAspectFlags` is a mask of zero or more VkImageAspectFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkClearAttachment, VkImageAspectFlagBits, VkImageSubresource, VkImageSubresourceLayers, VkImageSubresourceRange, VkSparseImageFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageAspectFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkImageCreateFlags(3)

Name

VkImageCreateFlags - Bitmask of VkImageCreateFlagBits

C Specification

```
typedef VkFlags VkImageCreateFlags;
```

Description

`VkImageCreateFlags` is a mask of zero or more VkImageCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkImageCreateFlagBits, VkImageCreateInfo, vkGetPhysicalDeviceImageFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkImageUsageFlags(3)

Name

VkImageUsageFlags - Bitmask of VkImageUsageFlagBits

C Specification

```
typedef VkFlags VkImageUsageFlags;
```

Description

`VkImageUsageFlags` is a mask of zero or more VkImageUsageFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkImageCreateInfo, VkImageUsageFlagBits, vkGetPhysicalDeviceImageFormatProperties, vkGetPhysicalDeviceSparseImageFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageUsageFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkImageViewCreateFlags(3)

Name

VkImageViewCreateFlags - Bitmask of VkImageViewCreateFlagBits

C Specification

```
typedef VkFlags VkImageViewCreateFlags;
```

Description

`VkImageViewCreateFlags` is a mask of zero or more VkImageViewCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkImageViewCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkImageViewCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkInstanceCreateFlags(3)

Name

VkInstanceCreateFlags - Bitmask of VkInstanceCreateFlagBits

C Specification

```
typedef VkFlags VkInstanceCreateFlags;
```

Description

`VkInstanceCreateFlags` is a mask of zero or more VkInstanceCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkInstanceCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkInstanceCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkMemoryHeapFlags(3)

Name

VkMemoryHeapFlags - Bitmask of VkMemoryHeapFlagBits

C Specification

```
typedef VkFlags VkMemoryHeapFlags;
```

Description

`VkMemoryHeapFlags` is a mask of zero or more VkMemoryHeapFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkMemoryHeap, VkMemoryHeapFlagBits

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkMemoryHeapFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkMemoryMapFlags(3)

Name

VkMemoryMapFlags - Bitmask of VkMemoryMapFlagBits

C Specification

```
typedef VkFlags VkMemoryMapFlags;
```

Description

`VkMemoryMapFlags` is a mask of zero or more VkMemoryMapFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

vkMapMemory

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkMemoryMapFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkMemoryPropertyFlags(3)

Name

VkMemoryPropertyFlags - Bitmask of VkMemoryPropertyFlagBits

C Specification

```
typedef VkFlags VkMemoryPropertyFlags;
```

Description

`VkMemoryPropertyFlags` is a mask of zero or more VkMemoryPropertyFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkMemoryPropertyFlagBits, VkMemoryType

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkMemoryPropertyFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineCacheCreateFlags(3)

Name

VkPipelineCacheCreateFlags - Bitmask of VkPipelineCacheCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineCacheCreateFlags;
```

Description

`VkPipelineCacheCreateFlags` is a mask of zero or more VkPipelineCacheCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineCacheCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineCacheCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineColorBlendStateCreateFlags(3)

Name

VkPipelineColorBlendStateCreateFlags - Bitmask of VkPipelineColorBlendStateCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineColorBlendStateCreateFlags;
```

Description

`VkPipelineColorBlendStateCreateFlags` is a mask of zero or more VkPipelineColorBlendStateCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineColorBlendStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineColorBlendStateCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineCreateFlags(3)

Name

VkPipelineCreateFlags - Bitmask of VkPipelineCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineCreateFlags;
```

Description

`VkPipelineCreateFlags` is a mask of zero or more VkPipelineCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkComputePipelineCreateInfo, VkGraphicsPipelineCreateInfo, VkPipelineCreateFlagBits

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineDepthStencilStateCreateFlags(3)

Name

VkPipelineDepthStencilStateCreateFlags - Bitmask of VkPipelineDepthStencilStateCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineDepthStencilStateCreateFlags;
```

Description

`VkPipelineDepthStencilStateCreateFlags` is a mask of zero or more VkPipelineDepthStencilStateCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineDepthStencilStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineDepthStencilStateCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineDynamicStateCreateFlags(3)

Name

VkPipelineDynamicStateCreateFlags - Bitmask of VkPipelineDynamicStateCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineDynamicStateCreateFlags;
```

Description

`VkPipelineDynamicStateCreateFlags` is a mask of zero or more VkPipelineDynamicStateCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineDynamicStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineDynamicStateCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineInputAssemblyStateCreateFlags(3)

Name

VkPipelineInputAssemblyStateCreateFlags - Bitmask of VkPipelineInputAssemblyStateCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineInputAssemblyStateCreateFlags;
```

Description

`VkPipelineInputAssemblyStateCreateFlags` is a mask of zero or more VkPipelineInputAssemblyStateCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineInputAssemblyStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineInputAssemblyStateCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineLayoutCreateFlags(3)

Name

VkPipelineLayoutCreateFlags - Bitmask of VkPipelineLayoutCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineLayoutCreateFlags;
```

Description

`VkPipelineLayoutCreateFlags` is a mask of zero or more VkPipelineLayoutCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineLayoutCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineLayoutCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineMultisampleStateCreateFlags(3)

Name

VkPipelineMultisampleStateCreateFlags - Bitmask of VkPipelineMultisampleStateCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineMultisampleStateCreateFlags;
```

Description

`VkPipelineMultisampleStateCreateFlags` is a mask of zero or more VkPipelineMultisampleStateCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineMultisampleStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineMultisampleStateCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineRasterizationStateCreateFlags(3)

Name

VkPipelineRasterizationStateCreateFlags - Bitmask of VkPipelineRasterizationStateCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineRasterizationStateCreateFlags;
```

Description

`VkPipelineRasterizationStateCreateFlags` is a mask of zero or more VkPipelineRasterizationStateCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineRasterizationStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineRasterizationStateCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineShaderStageCreateFlags(3)

Name

VkPipelineShaderStageCreateFlags - Bitmask of VkPipelineShaderStageCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineShaderStageCreateFlags;
```

Description

`VkPipelineShaderStageCreateFlags` is a mask of zero or more VkPipelineShaderStageCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineShaderStageCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineShaderStageCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineStageFlags(3)

Name

VkPipelineStageFlags - Bitmask of VkPipelineStageFlagBits

C Specification

```
typedef VkFlags VkPipelineStageFlags;
```

Description

`VkPipelineStageFlags` is a mask of zero or more VkPipelineStageFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineStageFlagBits, VkSubmitInfo, VkSubpassDependency, vkCmdPipelineBarrier, vkCmdResetEvent, vkCmdSetEvent, vkCmdWaitEvents

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineStageFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineTessellationStateCreateFlags(3)

Name

VkPipelineTessellationStateCreateFlags - Bitmask of VkPipelineTessellationStateCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineTessellationStateCreateFlags;
```

Description

`VkPipelineTessellationStateCreateFlags` is a mask of zero or more VkPipelineTessellationStateCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineTessellationStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineTessellationStateCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineVertexInputStateCreateFlags(3)

Name

VkPipelineVertexInputStateCreateFlags - Bitmask of VkPipelineVertexInputStateCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineVertexInputStateCreateFlags;
```

Description

`VkPipelineVertexInputStateCreateFlags` is a mask of zero or more VkPipelineVertexInputStateCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineVertexInputStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineVertexInputStateCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkPipelineViewportStateCreateFlags(3)

Name

VkPipelineViewportStateCreateFlags - Bitmask of VkPipelineViewportStateCreateFlagBits

C Specification

```
typedef VkFlags VkPipelineViewportStateCreateFlags;
```

Description

`VkPipelineViewportStateCreateFlags` is a mask of zero or more VkPipelineViewportStateCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkPipelineViewportStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkPipelineViewportStateCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkQueryControlFlags(3)

Name

VkQueryControlFlags - Bitmask of VkQueryControlFlagBits

C Specification

```
typedef VkFlags VkQueryControlFlags;
```

Description

`VkQueryControlFlags` is a mask of zero or more VkQueryControlFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkCommandBufferInheritanceInfo, VkQueryControlFlagBits, vkCmdBeginQuery

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueryControlFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkQueryPipelineStatisticFlags(3)

Name

VkQueryPipelineStatisticFlags - Bitmask of VkQueryPipelineStatisticFlagBits

C Specification

```
typedef VkFlags VkQueryPipelineStatisticFlags;
```

Description

`VkQueryPipelineStatisticFlags` is a mask of zero or more VkQueryPipelineStatisticFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkCommandBufferInheritanceInfo, VkQueryPipelineStatisticFlagBits, VkQueryPoolCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueryPipelineStatisticFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkQueryPoolCreateFlags(3)

Name

VkQueryPoolCreateFlags - Bitmask of VkQueryPoolCreateFlagBits

C Specification

```
typedef VkFlags VkQueryPoolCreateFlags;
```

Description

`VkQueryPoolCreateFlags` is a mask of zero or more VkQueryPoolCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkQueryPoolCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueryPoolCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkQueryResultFlags(3)

Name

VkQueryResultFlags - Bitmask of VkQueryResultFlagBits

C Specification

```
typedef VkFlags VkQueryResultFlags;
```

Description

`VkQueryResultFlags` is a mask of zero or more VkQueryResultFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkQueryResultFlagBits, vkCmdCopyQueryPoolResults, vkGetQueryPoolResults

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueryResultFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkQueueFlags(3)

Name

VkQueueFlags - Bitmask of VkQueueFlagBits

C Specification

```
typedef VkFlags VkQueueFlags;
```

Description

`VkQueueFlags` is a mask of zero or more VkQueueFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkQueueFamilyProperties, VkQueueFlagBits

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkQueueFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkRenderPassCreateFlags(3)

Name

VkRenderPassCreateFlags - Bitmask of VkRenderPassCreateFlagBits

C Specification

```
typedef VkFlags VkRenderPassCreateFlags;
```

Description

`VkRenderPassCreateFlags` is a mask of zero or more VkRenderPassCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkRenderPassCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkRenderPassCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkSampleCountFlags(3)

Name

VkSampleCountFlags - Bitmask of VkSampleCountFlagBits

C Specification

```
typedef VkFlags VkSampleCountFlags;
```

Description

`VkSampleCountFlags` is a mask of zero or more VkSampleCountFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkImageFormatProperties, VkPhysicalDeviceLimits, VkSampleCountFlagBits

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSampleCountFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkSamplerCreateFlags(3)

Name

VkSamplerCreateFlags - Bitmask of VkSamplerCreateFlagBits

C Specification

```
typedef VkFlags VkSamplerCreateFlags;
```

Description

`VkSamplerCreateFlags` is a mask of zero or more VkSamplerCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkSamplerCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSamplerCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkSemaphoreCreateFlags(3)

Name

VkSemaphoreCreateFlags - Bitmask of VkSemaphoreCreateFlagBits

C Specification

```
typedef VkFlags VkSemaphoreCreateFlags;
```

Description

`VkSemaphoreCreateFlags` is a mask of zero or more VkSemaphoreCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkSemaphoreCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSemaphoreCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkShaderModuleCreateFlags(3)

Name

VkShaderModuleCreateFlags - Bitmask of VkShaderModuleCreateFlagBits

C Specification

```
typedef VkFlags VkShaderModuleCreateFlags;
```

Description

`VkShaderModuleCreateFlags` is a mask of zero or more VkShaderModuleCreateFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkShaderModuleCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkShaderModuleCreateFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkShaderStageFlags(3)

Name

VkShaderStageFlags - Bitmask of VkShaderStageFlagBits

C Specification

```
typedef VkFlags VkShaderStageFlags;
```

Description

`VkShaderStageFlags` is a mask of zero or more VkShaderStageFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkDescriptorSetLayoutBinding, VkPushConstantRange, VkShaderStageFlagBits, vkCmdPushConstants

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkShaderStageFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkSparseImageFormatFlags(3)

Name

VkSparseImageFormatFlags - Bitmask of VkSparseImageFormatFlagBits

C Specification

```
typedef VkFlags VkSparseImageFormatFlags;
```

Description

`VkSparseImageFormatFlags` is a mask of zero or more VkSparseImageFormatFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkSparseImageFormatFlagBits, VkSparseImageFormatProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSparseImageFormatFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkSparseMemoryBindFlags(3)

Name

VkSparseMemoryBindFlags - Bitmask of VkSparseMemoryBindFlagBits

C Specification

```
typedef VkFlags VkSparseMemoryBindFlags;
```

Description

`VkSparseMemoryBindFlags` is a mask of zero or more VkSparseMemoryBindFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkSparseImageMemoryBind, VkSparseMemoryBind, VkSparseMemoryBindFlagBits

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSparseMemoryBindFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkStencilFaceFlags(3)

Name

VkStencilFaceFlags - Bitmask of VkStencilFaceFlagBits

C Specification

```
typedef VkFlags VkStencilFaceFlags;
```

Description

`VkStencilFaceFlags` is a mask of zero or more VkStencilFaceFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkStencilFaceFlagBits, vkCmdSetStencilCompareMask, vkCmdSetStencilReference, vkCmdSetStencilWriteMask

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkStencilFaceFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

VkSubpassDescriptionFlags(3)

Name

VkSubpassDescriptionFlags - Bitmask of VkSubpassDescriptionFlagBits

C Specification

```
typedef VkFlags VkSubpassDescriptionFlags;
```

Description

`VkSubpassDescriptionFlags` is a mask of zero or more VkSubpassDescriptionFlagBits. It is used as a member and/or parameter of the structures and commands in the See Also section below.

See Also

VkSubpassDescription, VkSubpassDescriptionFlagBits

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSubpassDescriptionFlags

This page is a generated document. Fixes and changes should be made to the generator scripts, not directly.

Function Pointer Types

PFN_vkAllocationFunction(3)

Name

PFN_vkAllocationFunction - Application-defined memory allocation function

C Specification

The type of pfnAllocation is:

```
typedef void* (VKAPI_PTR *PFN_vkAllocationFunction)(
    void*                                       pUserData,
    size_t                                      size,
    size_t                                      alignment,
    VkSystemAllocationScope                     allocationScope);
```

Parameters

- pUserData is the value specified for VkAllocationCallbacks::pUserData in the allocator specified by the application.
- size is the size in bytes of the requested allocation.
- alignment is the requested alignment of the allocation in bytes and **must** be a power of two.
- allocationScope is a VkSystemAllocationScope value specifying the allocation scope of the lifetime of the allocation, as described here.

Description

If pfnAllocation is unable to allocate the requested memory, it **must** return NULL. If the allocation was successful, it **must** return a valid pointer to memory allocation containing at least size bytes, and with the pointer value being a multiple of alignment.

> *Note*
>
> Correct Vulkan operation **cannot** be assumed if the application does not follow these rules.
>
> For example, pfnAllocation (or pfnReallocation) could cause termination of running Vulkan instance(s) on a failed allocation for debugging purposes, either directly or indirectly. In these circumstances, it **cannot** be assumed that any part of any affected VkInstance objects are going to operate correctly (even vkDestroyInstance), and the application **must** ensure it cleans up properly via other means (e.g. process termination).

If pfnAllocation returns NULL, and if the implementation is unable to continue correct processing of the current command without the requested allocation, it **must** treat this as a run-time error, and generate `VK_ERROR_OUT_OF_HOST_MEMORY` at the appropriate time for the command in which the condition was detected, as described in Return Codes.

If the implementation is able to continue correct processing of the current command without the requested allocation, then it **may** do so, and **must** not generate `VK_ERROR_OUT_OF_HOST_MEMORY` as a result of this failed allocation.

See Also

VkAllocationCallbacks

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#PFN_vkAllocationFunction

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

PFN_vkFreeFunction(3)

Name

PFN_vkFreeFunction - Application-defined memory free function

C Specification

The type of `pfnFree` is:

```
typedef void (VKAPI_PTR *PFN_vkFreeFunction)(
    void*                                       pUserData,
    void*                                       pMemory);
```

Parameters

- `pUserData` is the value specified for VkAllocationCallbacks::`pUserData` in the allocator specified by the application.
- `pMemory` is the allocation to be freed.

Description

`pMemory` **may** be `NULL`, which the callback **must** handle safely. If `pMemory` is non-`NULL`, it **must** be a pointer previously allocated by `pfnAllocation` or `pfnReallocation`. The application **should** free this memory.

See Also

VkAllocationCallbacks

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#PFN_vkFreeFunction

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

PFN_vkInternalAllocationNotification(3)

Name

PFN_vkInternalAllocationNotification - Application-defined memory allocation notification function

C Specification

The type of pfnInternalAllocation is:

```
typedef void (VKAPI_PTR *PFN_vkInternalAllocationNotification)(
    void*                                       pUserData,
    size_t                                      size,
    VkInternalAllocationType                    allocationType,
    VkSystemAllocationScope                     allocationScope);
```

Parameters

- pUserData is the value specified for VkAllocationCallbacks::pUserData in the allocator specified by the application.
- size is the requested size of an allocation.
- allocationType is a VkInternalAllocationType value specifying the requested type of an allocation.
- allocationScope is a VkSystemAllocationScope value specifying the allocation scope of the lifetime of the allocation, as described here.

Description

This is a purely informational callback.

See Also

VkAllocationCallbacks

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#PFN_vkInternalAllocationNotification

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

PFN_vkInternalFreeNotification(3)

Name

PFN_vkInternalFreeNotification - Application-defined memory free notification function

C Specification

The type of `pfnInternalFree` is:

```
typedef void (VKAPI_PTR *PFN_vkInternalFreeNotification)(
    void*                                       pUserData,
    size_t                                      size,
    VkInternalAllocationType                    allocationType,
    VkSystemAllocationScope                     allocationScope);
```

Parameters

- `pUserData` is the value specified for VkAllocationCallbacks::`pUserData` in the allocator specified by the application.
- `size` is the requested size of an allocation.
- `allocationType` is a VkInternalAllocationType value specifying the requested type of an allocation.
- `allocationScope` is a VkSystemAllocationScope value specifying the allocation scope of the lifetime of the allocation, as described here.

Description

See Also

VkAllocationCallbacks

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#PFN_vkInternalFreeNotification

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

PFN_vkReallocationFunction(3)

Name

PFN_vkReallocationFunction - Application-defined memory reallocation function

C Specification

The type of `pfnReallocation` is:

```
typedef void* (VKAPI_PTR *PFN_vkReallocationFunction)(
    void*                                       pUserData,
    void*                                       pOriginal,
    size_t                                      size,
    size_t                                      alignment,
    VkSystemAllocationScope                     allocationScope);
```

Parameters

- `pUserData` is the value specified for VkAllocationCallbacks::`pUserData` in the allocator specified by the application.
- `pOriginal` **must** be either `NULL` or a pointer previously returned by `pfnReallocation` or `pfnAllocation` of the same allocator.
- `size` is the size in bytes of the requested allocation.
- `alignment` is the requested alignment of the allocation in bytes and **must** be a power of two.
- `allocationScope` is a VkSystemAllocationScope value specifying the allocation scope of the lifetime of the allocation, as described here.

Description

`pfnReallocation` **must** return an allocation with enough space for `size` bytes, and the contents of the original allocation from bytes zero to min(original size, new size) - 1 **must** be preserved in the returned allocation. If `size` is larger than the old size, the contents of the additional space are undefined. If satisfying these requirements involves creating a new allocation, then the old allocation **should** be freed.

If `pOriginal` is `NULL`, then `pfnReallocation` **must** behave equivalently to a call to PFN_vkAllocationFunction with the same parameter values (without `pOriginal`).

If `size` is zero, then `pfnReallocation` **must** behave equivalently to a call to PFN_vkFreeFunction with the same `pUserData` parameter value, and `pMemory` equal to `pOriginal`.

If `pOriginal` is non-`NULL`, the implementation **must** ensure that `alignment` is equal to the `alignment` used to originally allocate `pOriginal`.

If this function fails and `pOriginal` is non-`NULL` the application **must** not free the old allocation.

`pfnReallocation` **must** follow the same rules for return values as `PFN_vkAllocationFunction`.

See Also

VkAllocationCallbacks

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#PFN_vkReallocationFunction

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

PFN_vkVoidFunction(3)

Name

PFN_vkVoidFunction - Dummy function pointer type returned by queries

C Specification

The definition of PFN_vkVoidFunction is:

```
typedef void (VKAPI_PTR *PFN_vkVoidFunction)(void);
```

Parameters

Description

See Also

vkGetDeviceProcAddr, vkGetInstanceProcAddr

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#PFN_vkVoidFunction

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

Vulkan Scalar types

VkBool32(3)

Name

VkBool32 - Vulkan boolean type

C Specification

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

```
typedef uint32_t VkBool32;
```

Description

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

See Also

VkCommandBufferInheritanceInfo, VkPhysicalDeviceFeatures, VkPhysicalDeviceLimits, VkPhysicalDeviceSparseProperties, VkPipelineColorBlendAttachmentState, VkPipelineColorBlendStateCreateInfo, VkPipelineDepthStencilStateCreateInfo, VkPipelineInputAssemblyStateCreateInfo, VkPipelineMultisampleStateCreateInfo, VkPipelineRasterizationStateCreateInfo, VkSamplerCreateInfo, vkWaitForFences

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkBool32

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkDeviceSize(3)

Name

VkDeviceSize - Vulkan device memory size and offsets

C Specification

`VkDeviceSize` represents device memory size and offset values:

```
typedef uint64_t VkDeviceSize;
```

Description

See Also

VkBufferCopy, VkBufferCreateInfo, VkBufferImageCopy, VkBufferMemoryBarrier, VkBufferViewCreateInfo, VkDescriptorBufferInfo, VkImageFormatProperties, VkMappedMemoryRange, VkMemoryAllocateInfo, VkMemoryHeap, VkMemoryRequirements, VkPhysicalDeviceLimits, VkSparseImageMemoryBind, VkSparseImageMemoryRequirements, VkSparseMemoryBind, VkSubresourceLayout, vkBindBufferMemory, vkBindImageMemory, vkCmdBindIndexBuffer, vkCmdBindVertexBuffers, vkCmdCopyQueryPoolResults, vkCmdDispatchIndirect, vkCmdDrawIndexedIndirect, vkCmdDrawIndirect, vkCmdFillBuffer, vkCmdUpdateBuffer, vkGetDeviceMemoryCommitment, vkGetQueryPoolResults, vkMapMemory

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkDeviceSize

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkFlags(3)

Name

VkFlags - Vulkan bitmasks

C Specification

A collection of flags is represented by a bitmask using the type `VkFlags`:

```
typedef uint32_t VkFlags;
```

Description

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. These aliases are described in the Flag Types appendix of the Specification.

See Also

VkColorComponentFlags

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkFlags

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VkSampleMask(3)

Name

VkSampleMask - Mask of sample coverage information

C Specification

The elements of the sample mask array are of type `VkSampleMask`, each representing 32 bits of coverage information:

```
typedef uint32_t VkSampleMask;
```

Description

See Also

VkPipelineMultisampleStateCreateInfo

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VkSampleMask

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

C Macro Definitions

VK_API_VERSION(3)

Name

VK_API_VERSION - Deprecated version number macro

C Specification

`VK_API_VERSION` is now commented out of vulkan.h and **cannot** be used.

```
// DEPRECATED: This define has been removed. Specific version defines (e.g.
VK_API_VERSION_1_0), or the VK_MAKE_VERSION macro, should be used instead.
//#define VK_API_VERSION VK_MAKE_VERSION(1, 0, 0) // Patch version should always be
set to 0
```

Description

See Also

No cross-references are available

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VK_API_VERSION

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VK_API_VERSION_1_0(3)

Name

VK_API_VERSION_1_0 - Return API version number for Vulkan 1.0

C Specification

`VK_API_VERSION_1_0` returns the API version number for Vulkan 1.0. The patch version number in this macro will always be zero. The supported patch version for a physical device **can** be queried with vkGetPhysicalDeviceProperties.

```
// Vulkan 1.0 version number
#define VK_API_VERSION_1_0 VK_MAKE_VERSION(1, 0, 0)// Patch version should always be
set to 0
```

Description

See Also

vkCreateInstance, vkGetPhysicalDeviceProperties

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VK_API_VERSION_1_0

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VK_DEFINE_HANDLE(3)

Name

VK_DEFINE_HANDLE - Declare a dispatchable object handle

C Specification

`VK_DEFINE_HANDLE` defines a dispatchable handle type.

```
#define VK_DEFINE_HANDLE(object) typedef struct object##_T* object;
```

Description

- `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.

See Also

VkCommandBuffer, VkDevice, VkInstance, VkPhysicalDevice, VkQueue

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VK_DEFINE_HANDLE

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VK_DEFINE_NON_DISPATCHABLE_HANDLE(3)

Name

VK_DEFINE_NON_DISPATCHABLE_HANDLE - Declare a non-dispatchable object handle

C Specification

`VK_DEFINE_NON_DISPATCHABLE_HANDLE` defines a non-dispatchable handle type.

```
#if !defined(VK_DEFINE_NON_DISPATCHABLE_HANDLE)
#if defined(__LP64__) || defined(_WIN64) || (defined(__x86_64__) &&
!defined(__ILP32__) ) || defined(_M_X64) || defined(__ia64) || defined (_M_IA64) ||
defined(__aarch64__) || defined(__powerpc64__)
        #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
```

Description

- `object` is the name of the resulting C type.

Most Vulkan handle types, such as VkBuffer, are non-dispatchable.

> *Note*
>
> The vulkan.h header allows the `VK_DEFINE_NON_DISPATCHABLE_HANDLE` definition to be overridden by the application. If `VK_DEFINE_NON_DISPATCHABLE_HANDLE` is already defined when the vulkan.h header is compiled the default definition is 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. Behavior is undefined if the application defines a non-binary-compatible handle and **may** result in memory corruption or application termination. Binary compatibility is platform dependent so the application **must** be careful if it overrides the default `VK_DEFINE_NON_DISPATCHABLE_HANDLE` definition.

See Also

VkBuffer

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#

VK_DEFINE_NON_DISPATCHABLE_HANDLE

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VK_HEADER_VERSION(3)

Name

VK_HEADER_VERSION - Vulkan header file version number

C Specification

`VK_HEADER_VERSION` is the version number of the vulkan.h header. This value is currently kept synchronized with the release number of the Specification. However, it is not guaranteed to remain synchronized, since most Specification updates have no effect on vulkan.h.

```
// Version of this file
#define VK_HEADER_VERSION 55
```

Description

See Also

No cross-references are available

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VK_HEADER_VERSION

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VK_MAKE_VERSION(3)

Name

VK_MAKE_VERSION - Construct an API version number

C Specification

`VK_MAKE_VERSION` constructs an API version number.

```
#define VK_MAKE_VERSION(major, minor, patch) \
    (((major) << 22) | ((minor) << 12) | (patch))
```

Description

- `major` is the major version number.
- `minor` is the minor version number.
- `patch` is the patch version number.

This macro **can** be used when constructing the VkApplicationInfo::`apiVersion` parameter passed to vkCreateInstance.

See Also

VkApplicationInfo, vkCreateInstance

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VK_MAKE_VERSION

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VK_NULL_HANDLE(3)

Name

VK_NULL_HANDLE - Reserved non-valid object handle

C Specification

`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.

```
#define VK_NULL_HANDLE 0
```

Description

See Also

No cross-references are available

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VK_NULL_HANDLE

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VK_VERSION_MAJOR(3)

Name

VK_VERSION_MAJOR - Extract API major version number

C Specification

`VK_VERSION_MAJOR` extracts the API major version number from a packed version number:

```
#define VK_VERSION_MAJOR(version) ((uint32_t)(version) >> 22)
```

Description

See Also

No cross-references are available

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VK_VERSION_MAJOR

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VK_VERSION_MINOR(3)

Name

VK_VERSION_MINOR - Extract API minor version number

C Specification

`VK_VERSION_MINOR` extracts the API minor version number from a packed version number:

```
#define VK_VERSION_MINOR(version) (((uint32_t)(version) >> 12) & 0x3ff)
```

Description

See Also

No cross-references are available

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VK_VERSION_MINOR

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.

VK_VERSION_PATCH(3)

Name

VK_VERSION_PATCH - Extract API patch version number

C Specification

`VK_VERSION_PATCH` extracts the API patch version number from a packed version number:

```
#define VK_VERSION_PATCH(version) ((uint32_t)(version) & 0xfff)
```

Description

See Also

No cross-references are available

Document Notes

For more information, see the Vulkan Specification at URL

https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html#VK_VERSION_PATCH

This page is extracted from the Vulkan Specification. Fixes and changes should be made to the Specification, not directly.