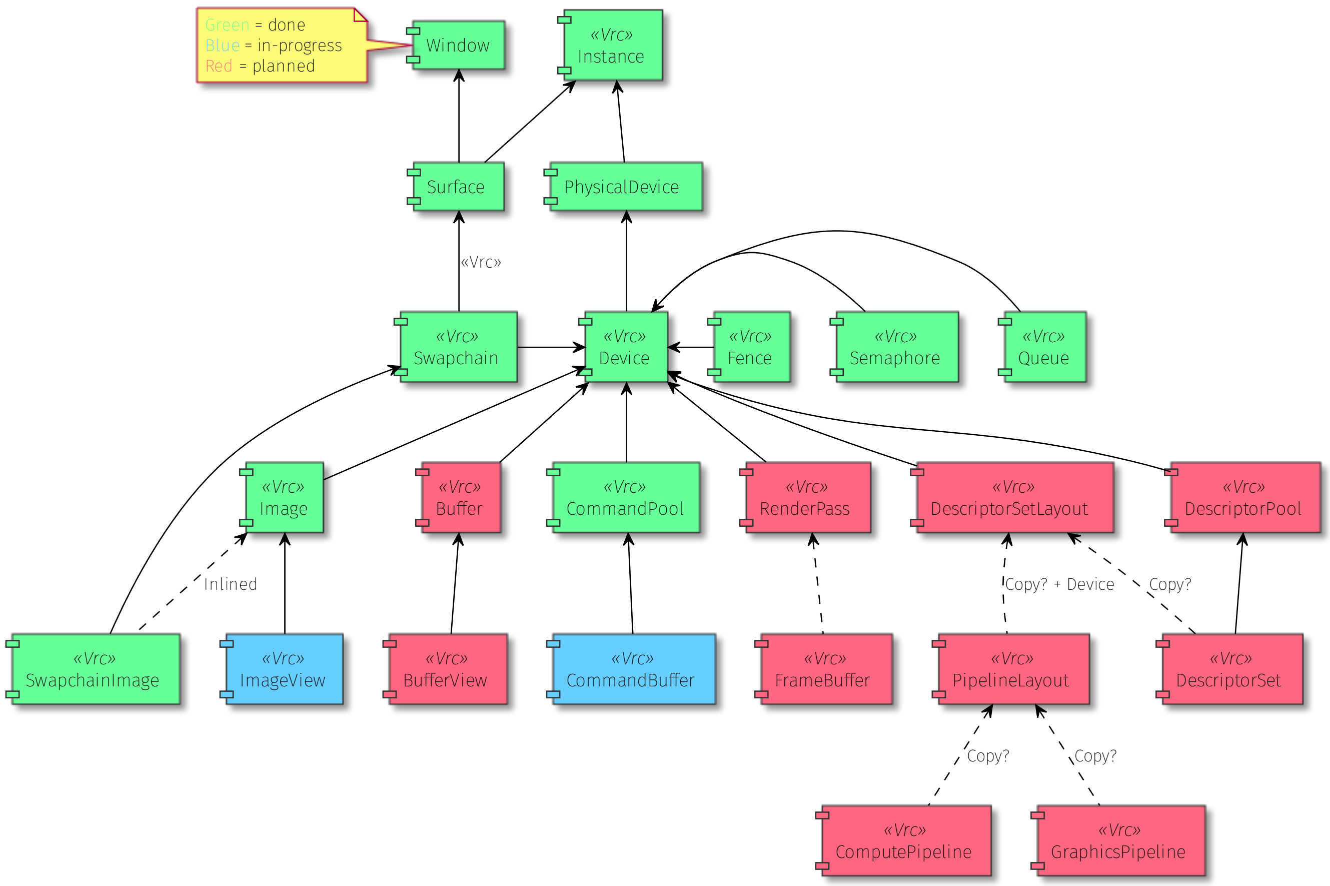

Figure 1: Object Dependency Graph

This document describes the plan and progress of the implementation of Vulkayes.

Synchronization

Most parameters in Vulkan require external synchronization. Synchronization is provided in two flavours: Single-thread and multi-thread. Single-thread synchronization primitives are noops, while multi-thread primitives provide actual multi-thread and multi-core synchronization. If single-thread synchronization is chosen, the Rust type system statically prevents use in multiple threads.

Externally Synchronized Parameters

- The `instance` parameter in `vkDestroyInstance`
    - Consequence of shared pointer usage
- The `device` parameter in `vkDestroyDevice`
    - Consequence of shared pointer usage
- The `queue` parameter in `vkQueueSubmit`
    - Synchronized internally
- The `fence` parameter in `vkQueueSubmit`
    - Synchronized internally
- The `queue` parameter in `vkQueueWaitIdle`
    - Synchronized internally
- The `memory` parameter in `vkFreeMemory`
    - Consequence of shared pointer usage
- The `memory` parameter in `vkMapMemory`
- The `memory` parameter in `vkUnmapMemory`
- The `buffer` parameter in `vkBindBufferMemory`
- The `image` parameter in `vkBindImageMemory`
    - Handled by API design
- The `queue` parameter in `vkQueueBindSparse`
- The `fence` parameter in `vkQueueBindSparse`
- The `fence` parameter in `vkDestroyFence`
    - Consequence of shared pointer usage
- The `semaphore` parameter in `vkDestroySemaphore`
    - Consequence of shared pointer usage
- The `event` parameter in `vkDestroyEvent`
- The `event` parameter in `vkSetEvent`
- The `event` parameter in `vkResetEvent`
- The `queryPool` parameter in `vkDestroyQueryPool`
- The `buffer` parameter in `vkDestroyBuffer`
- The `bufferView` parameter in `vkDestroyBufferView`
- The `image` parameter in `vkDestroyImage`
    - Consequence of shared pointer usage
- The `imageView` parameter in `vkDestroyImageView`
- The `shaderModule` parameter in `vkDestroyShaderModule`
- The `pipelineCache` parameter in `vkDestroyPipelineCache`
- The `dstCache` parameter in `vkMergePipelineCaches`
- The `pipeline` parameter in `vkDestroyPipeline`
- The `pipelineLayout` parameter in `vkDestroyPipelineLayout`
- The `sampler` parameter in `vkDestroySampler`
- The `descriptorSetLayout` parameter in `vkDestroyDescriptorSetLayout`
- The `descriptorPool` parameter in `vkDestroyDescriptorPool`
- The `descriptorPool` parameter in `vkResetDescriptorPool`
- The `descriptorPool` member of the `pAllocateInfo` parameter in `vkAllocateDescriptorSets`
- The `descriptorPool` parameter in `vkFreeDescriptorSets`
- The `framebuffer` parameter in `vkDestroyFramebuffer`
- The `renderPass` parameter in `vkDestroyRenderPass`

- The commandPool parameter in vkDestroyCommandPool
  - Consequence of shared pointer usage
- The commandPool parameter in vkResetCommandPool
  - Synchronized internally
- The commandPool member of the pAllocateInfo parameter in vkAllocateCommandBuffers
  - Synchronized internally
- The commandPool parameter in vkFreeCommandBuffers
  - Synchronized internally

- The commandBuffer parameter in vkBeginCommandBuffer
- The commandBuffer parameter in vkEndCommandBuffer
- The commandBuffer parameter in vkResetCommandBuffer
- The commandBuffer parameter in vkCmdBindPipeline
- The commandBuffer parameter in vkCmdSetViewport
- The commandBuffer parameter in vkCmdSetScissor
- The commandBuffer parameter in vkCmdSetLineWidth
- The commandBuffer parameter in vkCmdSetDepthBias
- The commandBuffer parameter in vkCmdSetBlendConstants
- The commandBuffer parameter in vkCmdSetDepthBounds
- The commandBuffer parameter in vkCmdSetStencilCompareMask
- The commandBuffer parameter in vkCmdSetStencilWriteMask
- The commandBuffer parameter in vkCmdSetStencilReference
- The commandBuffer parameter in vkCmdBindDescriptorSets
- The commandBuffer parameter in vkCmdBindIndexBuffer
- The commandBuffer parameter in vkCmdBindVertexBuffers
- The commandBuffer parameter in vkCmdDraw
- The commandBuffer parameter in vkCmdDrawIndexed
- The commandBuffer parameter in vkCmdDrawIndirect
- The commandBuffer parameter in vkCmdDrawIndexedIndirect
- The commandBuffer parameter in vkCmdDispatch
- The commandBuffer parameter in vkCmdDispatchIndirect
- The commandBuffer parameter in vkCmdCopyBuffer
- The commandBuffer parameter in vkCmdCopyImage
- The commandBuffer parameter in vkCmdBlitImage
- The commandBuffer parameter in vkCmdCopyBufferToImage
- The commandBuffer parameter in vkCmdCopyImageToBuffer
- The commandBuffer parameter in vkCmdUpdateBuffer
- The commandBuffer parameter in vkCmdFillBuffer
- The commandBuffer parameter in vkCmdClearColorImage
- The commandBuffer parameter in vkCmdClearDepthStencilImage
- The commandBuffer parameter in vkCmdClearAttachments
- The commandBuffer parameter in vkCmdResolveImage
- The commandBuffer parameter in vkCmdSetEvent
- The commandBuffer parameter in vkCmdResetEvent
- The commandBuffer parameter in vkCmdWaitEvents
- The commandBuffer parameter in vkCmdPipelineBarrier
- The commandBuffer parameter in vkCmdBeginQuery
- The commandBuffer parameter in vkCmdEndQuery
- The commandBuffer parameter in vkCmdResetQueryPool
- The commandBuffer parameter in vkCmdWriteTimestamp
- The commandBuffer parameter in vkCmdCopyQueryPoolResults
- The commandBuffer parameter in vkCmdPushConstants
- The commandBuffer parameter in vkCmdBeginRenderPass
- The commandBuffer parameter in vkCmdNextSubpass
- The commandBuffer parameter in vkCmdEndRenderPass

- The commandBuffer parameter in vkCmdExecuteCommands
- The commandBuffer parameter in vkCmdSetDeviceMask
- The commandBuffer parameter in vkCmdDispatchBase
- The commandPool parameter in vkTrimCommandPool
  - Internally synchronized
- The ycbcrConversion parameter in vkDestroySamplerYcbcrConversion
- The descriptorUpdateTemplate parameter in vkDestroyDescriptorUpdateTemplate
- The descriptorSet parameter in vkUpdateDescriptorSetWithTemplate
- The commandBuffer parameter in vkCmdDrawIndirectCount
- The commandBuffer parameter in vkCmdDrawIndexedIndirectCount
- The commandBuffer parameter in vkCmdBeginRenderPass2
- The commandBuffer parameter in vkCmdNextSubpass2
- The commandBuffer parameter in vkCmdEndRenderPass2
- The surface parameter in vkDestroySurfaceKHR
  - Consequence of shared pointer usage
- The surface member of the pCreateInfo parameter in vkCreateSwapchainKHR
  -
- The oldSwapchain member of the pCreateInfo parameter in vkCreateSwapchainKHR
  - Internally synchronized
- The swapchain parameter in vkDestroySwapchainKHR
  - Consequence of shared pointer usage
- The swapchain parameter in vkAcquireNextImageKHR
  - Internally synchronized
- The semaphore parameter in vkAcquireNextImageKHR
  - Internally synchronized
- The fence parameter in vkAcquireNextImageKHR
  - Internally synchronized
- The queue parameter in vkQueuePresentKHR
  - Internally synchronized
- The surface parameter in vkGetDeviceGroupSurfacePresentModesKHR
- The surface parameter in vkGetPhysicalDevicePresentRectanglesKHR
- The display parameter in vkCreateDisplayModeKHR
- The mode parameter in vkGetDisplayPlaneCapabilitiesKHR
- The commandBuffer parameter in vkCmdSetDeviceMaskKHR
- The commandBuffer parameter in vkCmdDispatchBaseKHR
- The commandBuffer parameter in vkCmdPushDescriptorSetKHR
- The commandBuffer parameter in vkCmdPushDescriptorSetWithTemplateKHR
- The descriptorUpdateTemplate parameter in vkDestroyDescriptorUpdateTemplateKHR
- The descriptorSet parameter in vkUpdateDescriptorSetWithTemplateKHR
- The commandBuffer parameter in vkCmdBeginRenderPass2KHR
- The commandBuffer parameter in vkCmdNextSubpass2KHR
- The commandBuffer parameter in vkCmdEndRenderPass2KHR
- The swapchain parameter in vkGetSwapchainStatusKHR
- The ycbcrConversion parameter in vkDestroySamplerYcbcrConversionKHR
- The commandBuffer parameter in vkCmdDrawIndirectCountKHR
- The commandBuffer parameter in vkCmdDrawIndexedIndirectCountKHR
- The callback parameter in vkDestroyDebugReportCallbackEXT
- The object member of the pTagInfo parameter in vkDebugMarkerSetObjectTagEXT
- The object member of the pNameInfo parameter in vkDebugMarkerSetObjectNameEXT
- The commandBuffer parameter in vkCmdBindTransformFeedbackBuffersEXT
- The commandBuffer parameter in vkCmdBeginTransformFeedbackEXT
- The commandBuffer parameter in vkCmdEndTransformFeedbackEXT
- The commandBuffer parameter in vkCmdBeginQueryIndexedEXT
- The commandBuffer parameter in vkCmdEndQueryIndexedEXT

- The `commandBuffer` parameter in `vkCmdDrawIndirectByteCountEXT`
- The `commandBuffer` parameter in `vkCmdDrawIndirectCountAMD`
- The `commandBuffer` parameter in `vkCmdDrawIndexedIndirectCountAMD`
- The `commandBuffer` parameter in `vkCmdBeginConditionalRenderingEXT`
- The `commandBuffer` parameter in `vkCmdEndConditionalRenderingEXT`
- The `commandBuffer` parameter in `vkCmdProcessCommandsNVX`
- The `commandBuffer` parameter in `vkCmdReserveSpaceForCommandsNVX`
- The `objectTable` parameter in `vkDestroyObjectTableNVX`
- The `objectTable` parameter in `vkRegisterObjectsNVX`
- The `objectTable` parameter in `vkUnregisterObjectsNVX`
- The `commandBuffer` parameter in `vkCmdSetViewportWScalingNV`
- The `swapchain` parameter in `vkGetRefreshCycleDurationGOOGLE`
- The `swapchain` parameter in `vkGetPastPresentationTimingGOOGLE`
- The `commandBuffer` parameter in `vkCmdSetDiscardRectangleEXT`
- The `objectHandle` member of the `pNameInfo` parameter in `vkSetDebugUtilsObjectNameEXT`
- The `objectHandle` member of the `pTagInfo` parameter in `vkSetDebugUtilsObjectTagEXT`
- The `messenger` parameter in `vkDestroyDebugUtilsMessengerEXT`
- The `commandBuffer` parameter in `vkCmdSetSampleLocationsEXT`
- The `validationCache` parameter in `vkDestroyValidationCacheEXT`
- The `dstCache` parameter in `vkMergeValidationCachesEXT`
- The `commandBuffer` parameter in `vkCmdBindShadingRateImageNV`
- The `commandBuffer` parameter in `vkCmdSetViewportShadingRatePaletteNV`
- The `commandBuffer` parameter in `vkCmdSetCoarseSampleOrderNV`
- The `commandBuffer` parameter in `vkCmdWriteBufferMarkerAMD`
- The `commandBuffer` parameter in `vkCmdDrawMeshTasksNV`
- The `commandBuffer` parameter in `vkCmdDrawMeshTasksIndirectNV`
- The `commandBuffer` parameter in `vkCmdDrawMeshTasksIndirectCountNV`
- The `commandBuffer` parameter in `vkCmdSetExclusiveScissorNV`
- The `commandBuffer` parameter in `vkCmdSetLineStippleEXT`

Validations

There are two types of validations in Vulkan API: Implicit validations, which talk about technical aspects of the API usage, and explicit validations, which talk about semantical aspects. Vulkayes aims to solve all implicit validations in the core crate. External validations are not always trivial to solve, some of them are statically fulfilled using the type system or the API design, others are left to the user.

External validations resolved statically are enclosed in blue boxes below. Validations optionally checked at runtime are in green boxes.

Implicit validations

Instance

Validations for `vkCreateInstance`:

1. `pCreateInfo` must be a valid pointer to a valid `VkInstanceCreateInfo` structure
   - Handled by API design (ash)
2. If `pAllocator` is not `NULL`, `pAllocator` must be a valid pointer to a valid `VkAllocationCallbacks` structure
   - Handled by API design (ash)
3. `pInstance` must be a valid pointer to a `VkInstance` handle
   - Handled by API design (ash)

Validations for VkInstanceCreateInfo:

1. sType must be VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO
    - Handled by API design (ash)
2. Each pNext member of any structure (including this one) in the pNext chain must be either NULL or a pointer to a valid instance of VkDebugReportCallbackCreateInfoEXT, VkDebugUtilsMessengerCreateInfoEXT, VkValidationFeaturesEXT, or VkValidationFlagsEXT
    - Handled by API design (ash)
3. The sType value of each struct in the pNext chain must be unique
    - Handled by API design
4. flags must be 0
    - Handled by API design (ash)
5. If pApplicationInfo is not NULL, pApplicationInfo must be a valid pointer to a valid VkApplicationInfo structure
    - Handled by API design (ash)
6. If enabledLayerCount is not 0, ppEnabledLayerNames must be a valid pointer to an array of enabledLayerCount null-terminated UTF-8 strings
    - Returns error
7. If enabledExtensionCount is not 0, ppEnabledExtensionNames must be a valid pointer to an array of enabledExtensionCount null-terminated UTF-8 strings
    - Returns error

Device

Validations for vkCreateDevice:

1. physicalDevice must be a valid VkPhysicalDevice handle
    - Handled by API design (ash)
2. pCreateInfo must be a valid pointer to a valid VkDeviceCreateInfo structure
    - Handled by API design (ash)
3. If pAllocator is not NULL, pAllocator must be a valid pointer to a valid VkAllocationCallbacks structure
    - Handled by API design (ash)
4. pDevice must be a valid pointer to a VkDevice handle
    - Handled by API design (ash)

Validations for VkDeviceCreateInfo:

1. sType must be VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO
   - Handled by API design (ash)
2. Each pNext member of any structure (including this one) in the pNext chain must be either NULL or a pointer to a valid instance of VkDeviceGroupDeviceCreateInfo, VkDeviceMemoryOverallocationCreateInfoAMD, VkPhysicalDevice16BitStorageFeatures, VkPhysicalDevice8BitStorageFeatures, VkPhysicalDeviceASTCDecodeFeaturesEXT, VkPhysicalDeviceBlendOperationAdvancedFeaturesEXT, VkPhysicalDeviceBufferDeviceAddressFeatures, VkPhysicalDeviceBufferDeviceAddressFeaturesEXT, VkPhysicalDeviceCoherentMemoryFeaturesAMD, VkPhysicalDeviceComputeShaderDerivativesFeaturesNV, VkPhysicalDeviceConditionalRenderingFeaturesEXT, VkPhysicalDeviceCooperativeMatrixFeaturesNV, VkPhysicalDeviceCornerSampledImageFeaturesNV, VkPhysicalDeviceCoverageReductionModeFeaturesNV, VkPhysicalDeviceDedicatedAllocationImageAliasingFeaturesNV, VkPhysicalDeviceDepthClipEnableFeaturesEXT, VkPhysicalDeviceDescriptorIndexingFeatures, VkPhysicalDeviceExclusiveScissorFeaturesNV, VkPhysicalDeviceFeatures2, VkPhysicalDeviceFragmentDensityMapFeaturesEXT, VkPhysicalDeviceFragmentShaderBarycentricFeaturesNV, VkPhysicalDeviceFragmentShaderInterlockFeaturesEXT, VkPhysicalDeviceHostQueryResetFeatures, VkPhysicalDeviceImagelessFramebufferFeatures, VkPhysicalDeviceIndexTypeUint8FeaturesEXT, VkPhysicalDeviceInlineUniformBlockFeaturesEXT, VkPhysicalDeviceLineRasterizationFeaturesEXT, VkPhysicalDeviceMemoryPriorityFeaturesEXT, VkPhysicalDeviceMeshShaderFeaturesNV, VkPhysicalDeviceMultiviewFeatures, VkPhysicalDevicePerformanceQueryFeaturesKHR, VkPhysicalDevicePipelineExecutablePropertiesFeaturesKHR, VkPhysicalDeviceProtectedMemoryFeatures, VkPhysicalDeviceRepresentativeFragmentTestFeaturesNV, VkPhysicalDeviceSamplerYcbcrConversionFeatures, VkPhysicalDeviceScalarBlockLayoutFeatures, VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures, VkPhysicalDeviceShaderAtomicInt64Features, VkPhysicalDeviceShaderClockFeaturesKHR, VkPhysicalDeviceShaderDemoteToHelperInvocationFeaturesEXT, VkPhysicalDeviceShaderDrawParametersFeatures, VkPhysicalDeviceShaderFloat16Int8Features, VkPhysicalDeviceShaderImageFootprintFeaturesNV, VkPhysicalDeviceShaderIntegerFunctions2FeaturesINTEL, VkPhysicalDeviceShaderSMBuiltinsFeaturesNV, VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures, VkPhysicalDeviceShadingRateImageFeaturesNV, VkPhysicalDeviceSubgroupSizeControlFeaturesEXT, VkPhysicalDeviceTexelBufferAlignmentFeaturesEXT, VkPhysicalDeviceTextureCompressionASTCHDRFeaturesEXT, VkPhysicalDeviceTimelineSemaphoreFeatures, VkPhysicalDeviceTransformFeedbackFeaturesEXT, VkPhysicalDeviceUniformBufferStandardLayoutFeatures, VkPhysicalDeviceVariablePointersFeatures, VkPhysicalDeviceVertexAttributeDivisorFeaturesEXT, VkPhysicalDeviceVulkan11Features, VkPhysicalDeviceVulkan12Features, VkPhysicalDeviceVulkanMemoryModelFeatures, or VkPhysicalDeviceYcbcrImageArraysFeaturesEXT
   - Handled by API design (ash)
3. The sType value of each struct in the pNext chain must be unique
   - Handled by API design
4. flags must be 0
   - Handled by API design (ash)
5. pQueueCreateInfos must be a valid pointer to an array of queueCreateInfoCount valid VkDeviceQueueCreateInfo structures
   - Handled by API design (ash)
6. If enabledLayerCount is not 0, ppEnabledLayerNames must be a valid pointer to an array of enabledLayerCount null-terminated UTF-8 strings
   - Returns error
7. If enabledExtensionCount is not 0, ppEnabledExtensionNames must be a valid pointer to an array of enabledExtensionCount null-terminated UTF-8 strings
   - Returns error

8. If `pEnabledFeatures` is not `NULL`, `pEnabledFeatures` must be a valid pointer to a valid `VkPhysicalDeviceFeatures` structure
    - Handled by API design (ash)
9. `queueCreateInfoCount` must be greater than `0`
    - Returns error

Queue

Validations for `VkDeviceQueueCreateInfo`:

1. `sType` must be `VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO`
    - Handled by API design (ash)
2. `pNext` must be `NULL` or a pointer to a valid instance of `VkDeviceQueueGlobalPriorityCreateInfoEXT`
    - Handled by API design (ash)
3. The `sType` value of each struct in the `pNext` chain must be unique
    - Handled by API design
4. `flags` must be a valid combination of `VkDeviceQueueCreateFlagBits` values
    - Handled by API design (ash)
5. `pQueuePriorities` must be a valid pointer to an array of `queueCount` `float` values
    - Handled by API design (ash)
6. `queueCount` must be greater than `0`
    - Returns error

Validations for `vkGetDeviceQueue`:

1. `device` must be a valid `VkDevice` handle
    - Handled by API design
2. `pQueue` must be a valid pointer to a `VkQueue` handle
    - Handled by API design

Validations for `vkGetDeviceQueue2`:

1. `device` must be a valid `VkDevice` handle
    - Handled by API design
2. `pQueueInfo` must be a valid pointer to a valid `VkDeviceQueueInfo2` structure
    - Handled by API design
3. `pQueue` must be a valid pointer to a `VkQueue` handle
    - Handled by API design

Validations for `VkDeviceQueueInfo2`:

1. `sType` must be `VK_STRUCTURE_TYPE_DEVICE_QUEUE_INFO_2`
    - Handled by API design (ash)
2. `pNext` must be `NULL`
    - Handled by API design (ash)
3. `flags` must be a valid combination of `VkDeviceQueueCreateFlagBits` values
    - Handled by API design (ash)

Validations for `vkQueueSubmit`:

1. `queue` must be a valid `VkQueue` handle
    - Handled by API design
2. If `submitCount` is not `0`, `pSubmits` must be a valid pointer to an array of `submitCount` valid `VkSubmitInfo` structures
    - Handled by API design
3. If `fence` is not `VK_NULL_HANDLE`, `fence` must be a valid `VkFence` handle
    - Handled by API design

4. Both of `fence`, and `queue` that are valid handles of non-ignored parameters must have been created, allocated, or retrieved from the same `VkDevice`
    - Returns error

Validations for `VkSubmitInfo`:

1. `sType` must be `VK_STRUCTURE_TYPE_SUBMIT_INFO`
    - Handled by API design (ash)
2. Each `pNext` member of any structure (including this one) in the `pNext` chain must be either `NULL` or a pointer to a valid instance of `VkD3D12FenceSubmitInfoKHR`, `VkDeviceGroupSubmitInfo`, `VkPerformanceQuerySubmitInfoKHR`, `VkProtectedSubmitInfo`, `VkTimelineSemaphoreSubmitInfo`, `VkWin32KeyedMutexAcquireReleaseInfoKHR`, or `VkWin32KeyedMutexAcquireReleaseInfoNV`
    - Handled by API design (ash)
3. The `sType` value of each struct in the `pNext` chain must be unique
    - Handled by API design
4. If `waitSemaphoreCount` is not `0`, `pWaitSemaphores` must be a valid pointer to an array of `waitSemaphoreCount` valid `VkSemaphore` handles
    - Handled by API design (ash)
5. If `waitSemaphoreCount` is not `0`, `pWaitDstStageMask` must be a valid pointer to an array of `waitSemaphoreCount` valid combinations of `VkPipelineStageFlagBits` values
    - Handled by API design (ash)
6. Each element of `pWaitDstStageMask` must not be `0`
    - Handled by API design
7. If `commandBufferCount` is not `0`, `pCommandBuffers` must be a valid pointer to an array of `commandBufferCount` valid `VkCommandBuffer` handles
    - Handled by API design (ash)
8. If `signalSemaphoreCount` is not `0`, `pSignalSemaphores` must be a valid pointer to an array of `signalSemaphoreCount` valid `VkSemaphore` handles
    - Handled by API design (ash)

9. Each of the elements of `pCommandBuffers`, the elements of `pSignalSemaphores`, and the elements of `pWaitSemaphores` that are valid handles of non-ignored parameters must have been created, allocated, or retrieved from the same `VkDevice`
    - Returns error

Swapchain

Validations for `vkCreateSwapchainKHR`:

1. `device` must be a valid `VkDevice` handle
   - Handled by API design (ash)
2. `pCreateInfo` must be a valid pointer to a valid `VkSwapchainCreateInfoKHR` structure
   - Handled by API design (ash)
3. If `pAllocator` is not `NULL`, `pAllocator` must be a valid pointer to a valid `VkAllocationCallbacks` structure
   - Handled by API design (ash)
4. `pSwapchain` must be a valid pointer to a `VkSwapchainKHR` handle
   - Handled by API design (ash)

Validations for `VkSwapchainCreateInfoKHR`:

1. `sType` must be `VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR`
   - Handled by API design (ash)
2. Each `pNext` member of any structure (including this one) in the `pNext` chain must be either `NULL` or a pointer to a valid instance of `VkDeviceGroupSwapchainCreateInfoKHR`, `VkImageFormatListCreateInfo`, `VkSurfaceFullScreenExclusiveInfoEXT`, `VkSurfaceFullScreenExclusiveWin32InfoEXT`, `VkSwapchainCounterCreateInfoEXT`, or `VkSwapchainDisplayNativeHdrCreateInfoAMD`
   - Handled by API design (ash)
3. The `sType` value of each struct in the `pNext` chain must be unique
   - Handled by API design
4. `flags` must be a valid combination of `VkSwapchainCreateFlagBitsKHR` values
   - Handled by API design (ash)
5. `surface` must be a valid `VkSurfaceKHR` handle
   - Handled by API design (ash)
6. `imageFormat` must be a valid `VkFormat` value
   - Handled by API design (ash)
7. `imageColorSpace` must be a valid `VkColorSpaceKHR` value
   - Handled by API design (ash)
8. `imageUsage` must be a valid combination of `VkImageUsageFlagBits` values
   - Handled by API design (ash)
9. `imageUsage` must not be `0`
   - Returns error
10. `imageSharingMode` must be a valid `VkSharingMode` value
    - Handled by API design (ash)
11. `preTransform` must be a valid `VkSurfaceTransformFlagBitsKHR` value
    - Handled by API design (ash)
12. `compositeAlpha` must be a valid `VkCompositeAlphaFlagBitsKHR` value
    - Handled by API design (ash)
13. `presentMode` must be a valid `VkPresentModeKHR` value
    - Handled by API design (ash)
14. If `oldSwapchain` is not `VK_NULL_HANDLE`, `oldSwapchain` must be a valid `VkSwapchainKHR` handle
    - Handled by API design (ash)
15. If `oldSwapchain` is a valid handle, it must have been created, allocated, or retrieved from `surface`
    - Handled by API design
16. Both of `oldSwapchain`, and `surface` that are valid handles of non-ignored parameters must have been created, allocated, or retrieved from the same `VkInstance`
    - Handled by API design

Validations for `vkGetSwapchainImagesKHR`:

1. `device` must be a valid `VkDevice` handle
    - Handled by API design
2. `swapchain` must be a valid `VkSwapchainKHR` handle
    - Handled by API design
3. `pSwapchainImageCount` must be a valid pointer to a `uint32_t` value
    - Handled by API design (ash)
4. If the value referenced by `pSwapchainImageCount` is not `0`, and `pSwapchainImages` is not `NULL`, `pSwapchainImages` must be a valid pointer to an array of `pSwapchainImageCount` `VkImage` handles
    - Handled by API design (ash)
5. Both of `device`, and `swapchain` must have been created, allocated, or retrieved from the same `VkInstance`
    - Handled by API design

Validations for `vkQueuePresentKHR`:

1. `queue` must be a valid `VkQueue` handle
    - Handled by API design (ash)
2. `pPresentInfo` must be a valid pointer to a valid `VkPresentInfoKHR` structure
    - Handled by API design (ash)

Validations for `VkPresentInfoKHR`:

1. `sType` must be `VK_STRUCTURE_TYPE_PRESENT_INFO_KHR`
    - Handled by API design (ash)
2. Each `pNext` member of any structure (including this one) in the `pNext` chain must be either `NULL` or a pointer to a valid instance of `VkDeviceGroupPresentInfoKHR`, `VkDisplayPresentInfoKHR`, `VkPresentFrameTokenGGP`, `VkPresentRegionsKHR`, or `VkPresentTimesInfoGOOGLE`
    - Handled by API design (ash)
3. The `sType` value of each struct in the `pNext` chain must be unique
    - Handled by API design
4. If `waitSemaphoreCount` is not `0`, `pWaitSemaphores` must be a valid pointer to an array of `waitSemaphoreCount` valid `VkSemaphore` handles
    - Handled by API design (ash)
5. `pSwapchains` must be a valid pointer to an array of `swapchainCount` valid `VkSwapchainKHR` handles
    - Handled by API design (ash)
6. `pImageIndices` must be a valid pointer to an array of `swapchainCount` `uint32_t` values
    - Handled by API design (ash)
7. If `pResults` is not `NULL`, `pResults` must be a valid pointer to an array of `swapchainCount` `VkResult` values
    - Handled by API design (ash)
8. `swapchainCount` must be greater than `0`
    - Returns error
9. Both of the elements of `pSwapchains`, and the elements of `pWaitSemaphores` that are valid handles of non-ignored parameters must have been created, allocated, or retrieved from the same `VkInstance`
    - Returns error

Validations for `vkAcquireNextImageKHR`:

1. device must be a valid VkDevice handle
    - Handled by API design
2. swapchain must be a valid VkSwapchainKHR handle
    - Handled by API design
3. If semaphore is not VK_NULL_HANDLE, semaphore must be a valid VkSemaphore handle
    - Handled by API design
4. If fence is not VK_NULL_HANDLE, fence must be a valid VkFence handle
    - Handled by API design
5. pImageIndex must be a valid pointer to a uint32_t value
    - Handled by API design (ash)
6. If semaphore is a valid handle, it must have been created, allocated, or retrieved from device
    - Returns error
7. If fence is a valid handle, it must have been created, allocated, or retrieved from device
    - Returns error
8. Both of device, and swapchain that are valid handles of non-ignored parameters must have been created, allocated, or retrieved from the same VkInstance
    - Handled by API design

Command Buffer

Validations for vkCreateCommandPool:

1. device must be a valid VkDevice handle
    - Handled by API design
2. pCreateInfo must be a valid pointer to a valid VkCommandPoolCreateInfo structure
    - Handled by API design (ash)
3. If pAllocator is not NULL, pAllocator must be a valid pointer to a valid VkAllocationCallbacks structure
    - Handled by API design
4. pCommandPool must be a valid pointer to a VkCommandPool handle
    - Handled by API design (ash)

Validations for VkCommandPoolCreateInfo:

1. sType must be VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO
    - Handled by API design (ash)
2. pNext must be NULL
    - Handled by API design (ash)
3. flags must be a valid combination of VkCommandPoolCreateFlagBits values
    - Handled by API design (ash)

Validations for vkTrimCommandPool:

1. device must be a valid VkDevice handle
    - Handled by API design
2. commandPool must be a valid VkCommandPool handle
    - Handled by API design
3. flags must be 0
    - Handled by API design
4. commandPool must have been created, allocated, or retrieved from device
    - Handled by API design

Validations for vkResetCommandPool:

1. device must be a valid VkDevice handle
    - Handled by API design
2. commandPool must be a valid VkCommandPool handle
    - Handled by API design
3. flags must be a valid combination of VkCommandPoolResetFlagBits values
    - Handled by API design
4. commandPool must have been created, allocated, or retrieved from device
    - Handled by API design

Validations for VkCommandBufferAllocateInfo:

1. sType must be VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO
    - Handled by API design (ash)
2. pNext must be NULL
    - Handled by API design (ash)
3. commandPool must be a valid VkCommandPool handle
    - Handled by API design (ash)
4. level must be a valid VkCommandBufferLevel value
    - Handled by API design (ash)

Fence

Validations for vkCreateFence:

1. device must be a valid VkDevice handle
    - Handled by API design (ash)
2. pCreateInfo must be a valid pointer to a valid VkFenceCreateInfo structure
    - Handled by API design (ash)
3. If pAllocator is not NULL, pAllocator must be a valid pointer to a valid VkAllocationCallbacks structure
    - Handled by API design (ash)
4. pFence must be a valid pointer to a VkFence handle
    - Handled by API design (ash)

Validations for VkFenceCreateInfo:

1. sType must be VK_STRUCTURE_TYPE_FENCE_CREATE_INFO
    - Handled by API design (ash)
2. Each pNext member of any structure (including this one) in the pNext chain must be either NULL or a pointer to a valid instance of VkExportFenceCreateInfo or VkExportFenceWin32HandleInfoKHR
    - Handled by API design (ash)
3. The sType value of each struct in the pNext chain must be unique
    - Handled by API design
4. flags must be a valid combination of VkFenceCreateFlagBits values
    - Handled by API design (ash)

Validations for vkGetFenceStatus:

1. device must be a valid VkDevice handle
    - Handled by API design
2. fence must be a valid VkFence handle
    - Handled by API design
3. fence must have been created, allocated, or retrieved from device
    - Handled by API design

Validations for vkResetFences:

1. device must be a valid VkDevice handle
    - Handled by API design
2. pFences must be a valid pointer to an array of fenceCount valid VkFence handles
    - Handled by API design
3. fenceCount must be greater than 0
    - Handled by API design
4. Each element of pFences must have been created, allocated, or retrieved from device
    - Handled by API design

Validations for vkWaitForFences:

1. device must be a valid VkDevice handle
    - Handled by API design
2. pFences must be a valid pointer to an array of fenceCount valid VkFence handles
    - Handled by API design
3. fenceCount must be greater than 0
    - Handled by API design
4. Each element of pFences must have been created, allocated, or retrieved from device
    - Handled by API design

Sempahore

Validations for vkCreateSemaphore:

1. `device` must be a valid `VkDevice` handle
    - Handled by API design (ash)
2. `pCreateInfo` must be a valid pointer to a valid `VkSemaphoreCreateInfo` structure
    - Handled by API design (ash)
3. If `pAllocator` is not `NULL`, `pAllocator` must be a valid pointer to a valid `VkAllocationCallbacks` structure
    - Handled by API design (ash)
4. `pSemaphore` must be a valid pointer to a `VkSemaphore` handle
    - Handled by API design (ash)

Validations for `VkSemaphoreCreateInfo`:

1. `sType` must be `VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO`
    - Handled by API design (ash)
2. Each `pNext` member of any structure (including this one) in the `pNext` chain must be either `NULL` or a pointer to a valid instance of `VkExportSemaphoreCreateInfo`, `VkExportSemaphoreWin32HandleInfoKHR`, or `VkSemaphoreTypeCreateInfo`
    - Handled by API design (ash)
3. The `sType` value of each struct in the `pNext` chain must be unique
    - Handled by API design
4. `flags` must be `0`
    - Handled by API design (ash)

Validations for `VkSemaphoreTypeCreateInfo`:

1. `sType` must be `VK_STRUCTURE_TYPE_SEMAPHORE_TYPE_CREATE_INFO`
    - Handled by API design (ash)
2. `semaphoreType` must be a valid `VkSemaphoreType` value
    - Handled by API design (ash)

Image

Validations for `vkCreateImage`:

1. `device` must be a valid `VkDevice` handle
    - Handled by API design
2. `pCreateInfo` must be a valid pointer to a valid `VkImageCreateInfo` structure
    - Handled by API design (ash)
3. If `pAllocator` is not `NULL`, `pAllocator` must be a valid pointer to a valid `VkAllocationCallbacks` structure
    - Handled by API design
4. `pImage` must be a valid pointer to a `VkImage` handle
    - Handled by API design (ash)

Validations for `VkImageCreateInfo`:

1. `sType` must be `VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO`
   - Handled by API design (ash)
2. Each `pNext` member of any structure (including this one) in the `pNext` chain must be either `NULL` or a pointer to a valid instance of `VkDedicatedAllocationImageCreateInfoNV`, `VkExternalFormatANDROID`, `VkExternalMemoryImageCreateInfo`, `VkExternalMemoryImageCreateInfoNV`, `VkImageDrmFormatModifierExplicitCreateInfoEXT`, `VkImageDrmFormatModifierListCreateInfoEXT`, `VkImageFormatListCreateInfo`, `VkImageStencilUsageCreateInfo`, or `VkImageSwapchainCreateInfoKHR`
   - Handled by API design (ash)
3. The `sType` value of each struct in the `pNext` chain must be unique
   - Handled by API design
4. `flags` must be a valid combination of `VkImageCreateFlagBits` values
   - Handled by API design
5. `imageType` must be a valid `VkImageType` value
   - Handled by API design (ash)
6. `format` must be a valid `VkFormat` value
   - Handled by API design (ash)
7. `samples` must be a valid `VkSampleCountFlagBits` value
   - Handled by API design (ash)
8. `tiling` must be a valid `VkImageTiling` value
   - Handled by API design (ash)
9. `usage` must be a valid combination of `VkImageUsageFlagBits` values
   - Handled by API design

10. `usage` must not be `0`
    - Return error

11. `sharingMode` must be a valid `VkSharingMode` value
    - Handled by API design (ash)
12. `initialLayout` must be a valid `VkImageLayout` value
    - Handled by API design (ash)

Validations for `vkBindImageMemory`:

1. `device` must be a valid `VkDevice` handle
   - Handled by API design
2. `image` must be a valid `VkImage` handle
   - Handled by API design
3. `memory` must be a valid `VkDeviceMemory` handle
   - Handled by API design
4. `image` must have been created, allocated, or retrieved from `device`
   - Handled by API design

5. `memory` must have been created, allocated, or retrieved from `device`
   - Return error

Creation validation

Validations of correct usage in create functions as dictated by the Vulkan specification.

Instance

Validations for `vkCreateInstance`:

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

Device

Validations for `vkCreateDevice`:

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

Validations for `VkDeviceCreateInfo`:

1. The `queueFamilyIndex` member of each element of `pQueueCreateInfos` must be unique within `pQueueCreateInfos`, except that two members can share the same `queueFamilyIndex` if one is a protected-capable queue and one is not a protected-capable queue
2. If the `pNext` chain includes a `VkPhysicalDeviceFeatures2` structure, then `pEnabledFeatures` must be `NULL`
    - Handled by API design
3. `ppEnabledExtensionNames` must not contain `VK_AMD_negative_viewport_height`
4. `ppEnabledExtensionNames` must not contain both `VK_KHR_buffer_device_address` and `VK_EXT_buffer_device_address`

5. If the pNext chain includes a VkPhysicalDeviceVulkan11Features structure, then it must not include a VkPhysicalDevice16BitStorageFeatures, VkPhysicalDeviceMultiviewFeatures, VkPhysicalDeviceVariablePointersFeatures, VkPhysicalDeviceProtectedMemoryFeatures, VkPhysicalDeviceSamplerYcbcrConversionFeatures, or VkPhysicalDeviceShaderDrawParametersFeatures structure
   - Handled by API design
6. If the pNext chain includes a VkPhysicalDeviceVulkan12Features structure, then it must not include a VkPhysicalDevice8BitStorageFeatures, VkPhysicalDeviceShader-AtomicInt64Features, VkPhysicalDeviceShaderFloat16Int8Features, VkPhysicalDe-viceDescriptorIndexingFeatures, VkPhysicalDeviceScalarBlockLayoutFeatures, VkPhysicalDeviceImagelessFramebufferFeatures, VkPhysicalDeviceUniformBuffer-StandardLayoutFeatures, VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures, VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures, VkPhysicalDeviceHost-QueryResetFeatures, VkPhysicalDeviceTimelineSemaphoreFeatures, VkPhysicalDe-viceBufferDeviceAddressFeatures, or VkPhysicalDeviceVulkanMemoryModelFeatures structure
   - Handled by API design
7. If ppEnabledExtensions contains code:“VK_KHR_draw_indirect_count” and the pNext chain includes a VkPhysicalDeviceVulkan12Features structure, then VkPhysicalDeviceVulkan12Features::drawIndirectCount must be VK_TRUE
   - Handled by API design
8. If ppEnabledExtensions contains code:“VK_KHR_sampler_mirror_clamp_to_edge” and the pNext chain includes a VkPhysicalDeviceVulkan12Features structure, then VkPhysicalDeviceVulkan12Features::samplerMirrorClampToEdge must be VK_TRUE
   - Handled by API design
9. If ppEnabledExtensions contains code:“VK_EXT_descriptor_indexing” and the pNext chain includes a VkPhysicalDeviceVulkan12Features structure, then VkPhysicalDeviceVulkan12Features::descriptorIndexing must be VK_TRUE
   - Handled by API design
10. If ppEnabledExtensions contains code:“VK_EXT_sampler_filter_minmax” and the pNext chain includes a VkPhysicalDeviceVulkan12Features structure, then VkPhysicalDeviceVulkan12Features::samplerFilterMinmax must be VK_TRUE
    - Handled by API design
11. If ppEnabledExtensions contains code:“VK_EXT_shader_viewport_index_layer” and the pNext chain includes a VkPhysicalDeviceVulkan12Features structure, then VkPhysicalDeviceVulkan12Features::shaderOutputViewportIndex and VkPhysicalDeviceVulkan12Features::shaderOutputLayer must both be VK_TRUE
    - Handled by API design

Queue

Validations for VkDeviceQueueCreateInfo:

1. queueFamilyIndex must be less than pQueueFamilyPropertyCount returned by vkGetPhysicalDeviceQueueFamilyProperties
2. queueCount must be less than or equal to the queueCount member of the VkQueueFamilyProperties structure, as returned by vkGetPhysicalDeviceQueueFamilyProperties in the pQueueFamilyProperties[queueFamilyIndex]
3. Each element of pQueuePriorities must be between 0.0 and 1.0 inclusive

4. If the protected memory feature is not enabled, the VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT bit of flags must not be set.
    - Handled by API design

Swapchain

Validations for VkSwapchainCreateInfoKHR:

1. surface must be a surface that is supported by the device as determined using vkGetPhysicalDeviceSurfaceSupportKHR
2. minImageCount must be less than or equal to the value returned in the maxImageCount member of the VkSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR for the surface if the returned maxImageCount is not zero
3. If presentMode is not VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR nor VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR, then minImageCount must be greater than or equal to the value returned in the minImageCount member of the VkSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR for the surface
4. minImageCount must be 1 if presentMode is either VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR or VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR
5. imageFormat and imageColorSpace must match the format and colorSpace members, respectively, of one of the VkSurfaceFormatKHR structures returned by vkGetPhysicalDeviceSurfaceFormatsKHR for the surface
6. imageExtent must be between minImageExtent and maxImageExtent, inclusive, where minImageExtent and maxImageExtent are members of the VkSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR for the surface
7. imageExtent members width and height must both be non-zero
    - Guaranteed by the type system
8. imageArrayLayers must be greater than 0 and less than or equal to the maxImageArrayLayers member of the VkSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR for the surface
    - Lower bound guaranteed by the type system
9. If presentMode is VK_PRESENT_MODE_IMMEDIATE_KHR, VK_PRESENT_MODE_MAILBOX_KHR, VK_PRESENT_MODE_FIFO_KHR or VK_PRESENT_MODE_FIFO_RELAXED_KHR, imageUsage must be a subset of the supported usage flags present in the supportedUsageFlags member of the VkSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR for surface
10. If presentMode is VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR or VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR, imageUsage must be a subset of the supported usage flags present in the sharedPresentSupportedUsageFlags member of the VkSharedPresentSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilities2KHR for surface
11. If imageSharingMode is VK_SHARING_MODE_CONCURRENT, pQueueFamilyIndices must be a valid pointer to an array of queueFamilyIndexCount uint32_t values
    - Guaranteed by the type system
12. If imageSharingMode is VK_SHARING_MODE_CONCURRENT, queueFamilyIndexCount must be greater than 1
    - Guaranteed by the type system
13. If imageSharingMode is VK_SHARING_MODE_CONCURRENT, each element of pQueueFamilyIndices must be unique and must be less than pQueueFamilyPropertyCount

returned by either vkGetPhysicalDeviceQueueFamilyProperties or vkGetPhysicalDeviceQueueFamilyProperties2 for the physicalDevice that was used to create device

14. preTransform must be one of the bits present in the supportedTransforms member of the VkSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR for the surface
15. compositeAlpha must be one of the bits present in the supportedCompositeAlpha member of the VkSurfaceCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilitiesKHR for the surface
16. presentMode must be one of the VkPresentModeKHR values returned by vkGetPhysicalDeviceSurfacePresentModesKHR for the surface
17. If the logical device was created with VkDeviceGroupDeviceCreateInfo::physicalDeviceCount equal to 1, flags must not contain VK_SWAPCHAIN_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT_KHR
    - Handled by API design
18. If oldSwapchain is not VK_NULL_HANDLE, oldSwapchain must be a non-retired swapchain associated with native window referred to by surface
    - Handled by API design
19. The implied image creation parameters of the swapchain must be supported as reported by vkGetPhysicalDeviceImageFormatProperties
20. If flags contains VK_SWAPCHAIN_CREATE_MUTABLE_FORMAT_BIT_KHR then the pNext chain must include a VkImageFormatListCreateInfo structure with a viewFormatCount greater than zero and pViewFormats must have an element equal to imageFormat
    - Handled by API design
21. If flags contains VK_SWAPCHAIN_CREATE_PROTECTED_BIT_KHR, then VkSurfaceProtectedCapabilitiesKHR::supportsProtected must be VK_TRUE in the VkSurfaceProtectedCapabilitiesKHR structure returned by vkGetPhysicalDeviceSurfaceCapabilities2KHR for surface
    - Handled by API design
22. If the pNext chain includes a VkSurfaceFullScreenExclusiveInfoEXT structure with its fullScreenExclusive member set to VK_FULL_SCREEN_EXCLUSIVE_APPLICATION_CONTROLLED_EXT, and surface was created using vkCreateWin32SurfaceKHR, a VkSurfaceFullScreenExclusiveWin32InfoEXT structure must be included in the pNext chain
    - Handled by API design

Command buffer

Validations for vkCreateCommandPool:

1. pCreateInfo→queueFamilyIndex must be the index of a queue family available in the logical device device.
    - Handled by API design

Validations for VkCommandPoolCreateInfo:

1. If the protected memory feature is not enabled, the VK_COMMAND_POOL_CREATE_PROTECTED_BIT bit of flags must not be set.
    - Handled by API design

Validations for VkCommandBufferAllocateInfo:

1. `commandBufferCount` must be greater than `0`
    - Guaranteed by the type system

Image

Validations for `vkCreateImage`:

1. 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`

Validations for `VkImageCreateInfo`:

1. Each of the following values (as described in Image Creation Limits) must not be undefined `imageCreateMaxMipLevels`, `imageCreateMaxArrayLayers`, `imageCreateMaxExtent`, and `imageCreateSampleCounts`.

2. If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, `pQueueFamilyIndices` must be a valid pointer to an array of `queueFamilyIndexCount` `uint32_t` values
    - Handled by API design
3. If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, `queueFamilyIndexCount` must be greater than `1`
    - Handled by API design
4. If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, each element of `pQueueFamilyIndices` must be unique and must be less than `pQueueFamilyPropertyCount` returned by either `vkGetPhysicalDeviceQueueFamilyProperties` or `vkGetPhysicalDeviceQueueFamilyProperties2` for the `physicalDevice` that was used to create `device`
    - Lower bound handled by API design
5. If the `pNext` chain includes a `VkExternalFormatANDROID` structure, and its `externalFormat` member is non-zero the `format` must be `VK_FORMAT_UNDEFINED`.
    - Handled by API design
6. If the `pNext` chain does not include a `VkExternalFormatANDROID` structure, or does and its `externalFormat` member is `0`, the `format` must not be `VK_FORMAT_UNDEFINED`.
    - Handled by API design
7. `extent.width` must be greater than `0`.
    - Guaranteed by the type system
8. `extent.height` must be greater than `0`.
    - Guaranteed by the type system
9. `extent.depth` must be greater than `0`.
    - Guaranteed by the type system
10. `mipLevels` must be greater than `0`
    - Guaranteed by the type system
11. `arrayLayers` must be greater than `0`
    - Guaranteed by the type system
12. If `flags` contains `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`, `imageType` must be `VK_IMAGE_TYPE_2D`
    - Guaranteed by the type system
13. If `flags` contains `VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT`, `imageType` must be `VK_IMAGE_TYPE_2D`
    - Guaranteed by the type system
14. If `flags` contains `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT`, `imageType` must be `VK_IMAGE_TYPE_3D`
    - Guaranteed by the type system
15. `extent.width` must be less than or equal to `imageCreateMaxExtent.width` (as defined in Image Creation Limits).
16. `extent.height` must be less than or equal to `imageCreateMaxExtent.height` (as defined in Image Creation Limits).
17. `extent.depth` must be less than or equal to `imageCreateMaxExtent.depth` (as defined in Image Creation Limits).
18. 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
    - Guaranteed by the type system
19. If `imageType` is `VK_IMAGE_TYPE_1D`, both `extent.height` and `extent.depth` must be `1`
    - Guaranteed by the type system
20. If `imageType` is `VK_IMAGE_TYPE_2D`, `extent.depth` must be `1`
    - Guaranteed by the type system
21. `mipLevels` must be less than or equal to the number of levels in the complete mipmap chain based on `extent.width`, `extent.height`, and `extent.depth`.
    - Guaranteed by the type system

22. `mipLevels` must be less than or equal to `imageCreateMaxMipLevels` (as defined in Image Creation Limits).
23. `arrayLayers` must be less than or equal to `imageCreateMaxArrayLayers` (as defined in Image Creation Limits).
24. If `imageType` is `VK_IMAGE_TYPE_3D`, `arrayLayers` must be `1`.
    - Guaranteed by the type system
25. If `samples` is not `VK_SAMPLE_COUNT_1_BIT`, then `imageType` must be `VK_IMAGE_TYPE_2D`, `flags` must not contain `VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT`, `mipLevels` must be equal to `1`, and `imageCreateMaybeLinear` (as defined in Image Creation Limits) must be `false`,
    - Guaranteed by the type system
26. If `samples` is not `VK_SAMPLE_COUNT_1_BIT`, `usage` must not contain `VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT`
    - Guaranteed by the type system
27. 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
28. 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`
29. 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`
30. If `usage` includes `VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT`, `extent.width` must be less than or equal to $\left\lceil \frac{maxFramebufferWidth}{minFragmentDensityTexelSize_{width}} \right\rceil$
31. If `usage` includes `VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT`, `extent.height` must be less than or equal to $\left\lceil \frac{maxFramebufferHeight}{minFragmentDensityTexelSize_{height}} \right\rceil$
32. 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`.
33. `samples` must be a bit value that is set in `imageCreateSampleCounts` (as defined in Image Creation Limits).
34. 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`
35. If the sparse bindings feature is not enabled, `flags` must not contain `VK_IMAGE_CREATE_SPARSE_BINDING_BIT`
36. If the sparse aliased residency feature is not enabled, `flags` must not contain `VK_IMAGE_CREATE_SPARSE_ALIASED_BIT`
37. If `imageType` is `VK_IMAGE_TYPE_1D`, `flags` must not contain `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`
38. 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`
39. 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`
40. 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`

41. 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`
42. 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`
43. 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`
44. 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`
45. 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
46. If the protected memory feature is not enabled, `flags` must not contain `VK_IMAGE_CREATE_PROTECTED_BIT`.
47. 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_CREATE_PROTECTED_BIT` must not also be set.
48. If the `pNext` chain includes a `VkExternalMemoryImageCreateInfoNV` structure, it must not contain a `VkExternalMemoryImageCreateInfo` structure.
49. If the `pNext` chain includes a `VkExternalMemoryImageCreateInfo` structure, its `handleTypes` member must only contain bits that are also in `VkExternalImageFormatProperties::externalMemoryProperties.compatibleHandleTypes`, as returned by `vkGetPhysicalDeviceImageFormatProperties2` with `format`, `imageType`, `tiling`, `usage`, and `flags` equal to those in this structure, and with a `VkPhysicalDeviceExternalImageFormatInfo` structure included in the `pNext` chain, with a `handleType` equal to any one of the handle types specified in `VkExternalMemoryImageCreateInfo::handleTypes`
50. If the `pNext` chain includes a `VkExternalMemoryImageCreateInfoNV` structure, its `handleTypes` member must only contain bits that are also in `VkExternalImageFormatPropertiesNV::externalMemoryProperties.compatibleHandleTypes`, as returned by `vkGetPhysicalDeviceExternalImageFormatPropertiesNV` with `format`, `imageType`, `tiling`, `usage`, and `flags` equal to those in this structure, and with `externalHandleType` equal to any one of the handle types specified in `VkExternalMemoryImageCreateInfoNV::handleTypes`
51. If the logical device was created with `VkDeviceGroupDeviceCreateInfo::physicalDeviceCount` equal to 1, `flags` must not contain `VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT`
52. If `flags` contains `VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT`, then `mipLevels` must be one, `arrayLayers` must be one, `imageType` must be `VK_IMAGE_TYPE_2D`. and `imageCreateMaybeLinear` (as defined in Image Creation Limits) must be `false`.
53. If `flags` contains `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT`, then `format` must be a block-compressed image format, an ETC compressed image format, or an ASTC compressed image format.
54. If `flags` contains `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT`, then `flags` must also contain `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`.
55. `initialLayout` must be `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`.
    - Guaranteed by the type system
56. `If the `pNext` chain includes a `VkExternalMemoryImageCreateInfo` or `VkExternalMemoryImageC`
57. If the image `format` is one of those listed in Formats requiring sampler Y�CBCRconversion forVK_IMAGE_ASPECT_COLOR_BITimage views, then `mipLevels` must be 1

58. If the image `format` is one of those listed in Formats requiring sampler Y⍰CBCRconversion forVK_IMAGE_ASPECT_COLOR_BITimage views, `samples` must be `VK_SAMPLE_COUNT_1_BIT`
59. If the image `format` is one of those listed in Formats requiring sampler Y⍰CBCRconversion forVK_IMAGE_ASPECT_COLOR_BITimage views, `imageType` must be `VK_IMAGE_TYPE_2D`
60. If the image `format` is one of those listed in Formats requiring sampler Y⍰CBCRconversion forVK_IMAGE_ASPECT_COLOR_BITimage views, and the `ycbcrImageArrays` feature is not enabled, `arrayLayers` must be 1
61. If `format` is a *multi-planar* format, and if `imageCreateFormatFeatures` (as defined in Image Creation Limits) does not contain `VK_FORMAT_FEATURE_DISJOINT_BIT`, then `flags` must not contain `VK_IMAGE_CREATE_DISJOINT_BIT`
62. If `format` is not a *multi-planar* format, and `flags` does not include `VK_IMAGE_CREATE_ALIAS_BIT`, `flags` must not contain `VK_IMAGE_CREATE_DISJOINT_BIT`
63. If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then the `pNext` chain must include exactly one of `VkImageDrmFormatModifierListCreateInfoEXT` or `VkImageDrmFormatModifierExplicitCreateInfoEXT` structures
64. If the `pNext` chain includes a `VkImageDrmFormatModifierListCreateInfoEXT` or `VkImageDrmFormatModifierExplicitCreateInfoEXT` structure, then `tiling` must be `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`
65. If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT` and `flags` contains `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`, then the `pNext` chain must include a `VkImageFormatListCreateInfo` structure with non-zero `viewFormatCount`.
66. If `flags` contains `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` `format` must be a depth or depth/stencil format
67. If the `pNext` chain includes a `VkExternalMemoryImageCreateInfo` structure whose `handleTypes` member includes `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`, `imageType` must be `VK_IMAGE_TYPE_2D`.
68. If the `pNext` chain includes a `VkExternalMemoryImageCreateInfo` structure whose `handleTypes` member includes `VK_EXTERNAL_MEMORY_HANDLE_TYPE_ANDROID_HARDWARE_BUFFER_BIT_ANDROID`, `mipLevels` must either be `1` or equal to the number of levels in the complete mipmap chain based on `extent.width`, `extent.height`, and `extent.depth`.
69. If the `pNext` chain includes a `VkExternalFormatANDROID` structure whose `externalFormat` member is not `0`, `flags` must not include `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`.
70. If the `pNext` chain includes a `VkExternalFormatANDROID` structure whose `externalFormat` member is not `0`, `usage` must not include any usages except `VK_IMAGE_USAGE_SAMPLED_BIT`.
71. If the `pNext` chain includes a `VkExternalFormatANDROID` structure whose `externalFormat` member is not `0`, `tiling` must be `VK_IMAGE_TILING_OPTIMAL`.
72. If `format` is a depth-stencil format, `usage` includes `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, and the `pNext` chain includes a `VkImageStencilUsageCreateInfo` structure, then its `VkImageStencilUsageCreateInfo`::`stencilUsage` member must also include `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
73. If `format` is a depth-stencil format, `usage` does not include `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, and the `pNext` chain includes a `VkImageStencilUsageCreateInfo` structure, then its `VkImageStencilUsageCreateInfo`::`stencilUsage` member must also not include `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
74. If `format` is a depth-stencil format, `usage` includes `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`, and the `pNext` chain includes a `VkImageStencilUsageCreateInfo` structure, then its `VkImageStencilUsageCreateInfo`::`stencilUsage` member must also include `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`
75. If `format` is a depth-stencil format, `usage` does not include `VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT`, and the `pNext` chain includes a

VkImageStencilUsageCreateInfo structure, then its VkImageStencilUsageCreateInfo::stencilUsage member must also not include VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT

76. If Format is a depth-stencil format and the pNext chain includes a VkImageStencilUsageCreateInfo structure with its stencilUsage member including VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT, extent.width must be less than or equal to VkPhysicalDeviceLimits::maxFramebufferWidth
77. If format is a depth-stencil format and the pNext chain includes a VkImageStencilUsageCreateInfo structure with its stencilUsage member including VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT, extent.height must be less than or equal to VkPhysicalDeviceLimits::maxFramebufferHeight
78. If the multisampled storage images feature is not enabled, format is a depth-stencil format and the pNext chain includes a VkImageStencilUsageCreateInfo structure with its stencilUsage including VK_IMAGE_USAGE_STORAGE_BIT, samples must be VK_SAMPLE_COUNT_1_BIT
79. If flags contains VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV, imageType must be VK_IMAGE_TYPE_2D or VK_IMAGE_TYPE_3D
80. If flags contains VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV, it must not contain VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT and the format must not be a depth/stencil format
81. If flags contains VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV and imageType is VK_IMAGE_TYPE_2D, extent.width and extent.height must be greater than 1
82. If flags contains VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV and imageType is VK_IMAGE_TYPE_3D, extent.width, extent.height, and extent.depth must be greater than 1
83. If usage includes VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV, imageType must be VK_IMAGE_TYPE_2D.
84. If usage includes VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV, samples must be VK_SAMPLE_COUNT_1_BIT.
85. If usage includes VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV, tiling must be VK_IMAGE_TILING_OPTIMAL.
86. If flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT, tiling must be VK_IMAGE_TILING_OPTIMAL
87. If flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT, imageType must be VK_IMAGE_TYPE_2D
88. If flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT, flags must not contain VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT
89. If flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT, mipLevels must be 1

Validations for VkImageViewCreateInfo:

1. 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
2. If the image cubemap arrays feature is not enabled, viewType must not be VK_IMAGE_VIEW_TYPE_CUBE_ARRAY
3. If image was created with VK_IMAGE_TYPE_3D but without VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT set then viewType must not be VK_IMAGE_VIEW_TYPE_2D or VK_IMAGE_VIEW_TYPE_2D_ARRAY
4. 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, VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT, VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV, or VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT
5. The format features of the resultant image view must contain at least one bit.
6. If usage contains VK_IMAGE_USAGE_SAMPLED_BIT, then the format features of the resultant

image view must contain `VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT`.

7. If `usage` contains `VK_IMAGE_USAGE_STORAGE_BIT`, then the image view's format features must contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT`.
8. If `usage` contains `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`, then the image view's format features must contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`.
9. If `usage` contains `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`, then the image view's format features must contain `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT`.
10. If `usage` contains `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`, then the image view's format features must contain at least one of `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT` or `VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT`.
11. `subresourceRange.baseMipLevel` must be less than the `mipLevels` specified in `VkImageCreateInfo` when `image` was created
12. If `subresourceRange.levelCount` is not `VK_REMAINING_MIP_LEVELS`, `subresourceRange.baseMipLevel+subresourceRange.levelCount` must be less than or equal to the `mipLevels` specified in `VkImageCreateInfo` when `image` was created
13. If `image` was created with `usage` containing `VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT`, `subresourceRange.levelCount` must be `1`
14. If `image` is not a 3D image created with `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT` set, or `viewType` is not `VK_IMAGE_VIEW_TYPE_2D` or `VK_IMAGE_VIEW_TYPE_2D_ARRAY`, `subresourceRange.baseArrayLayer` must be less than the `arrayLayers` specified in `VkImageCreateInfo` when `image` was created
15. If `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `image` is not a 3D image created with `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT` set, or `viewType` is not `VK_IMAGE_VIEW_TYPE_2D` or `VK_IMAGE_VIEW_TYPE_2D_ARRAY`, `subresourceRange.layerCount` must be non-zero and `subresourceRange.baseArrayLayer+subresourceRange.layerCount` must be less than or equal to the `arrayLayers` specified in `VkImageCreateInfo` when `image` was created
16. If `image` is a 3D image created with `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT` set, and `viewType` is `VK_IMAGE_VIEW_TYPE_2D` or `VK_IMAGE_VIEW_TYPE_2D_ARRAY`, `subresourceRange.baseArrayLayer` must be less than the depth computed from `baseMipLevel` and `extent.depth` specified in `VkImageCreateInfo` when `image` was created, according to the formula defined in Image Miplevel Sizing.
17. If `subresourceRange.layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, `image` is a 3D image created with `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT` set, and `viewType` is `VK_IMAGE_VIEW_TYPE_2D` or `VK_IMAGE_VIEW_TYPE_2D_ARRAY`, `subresourceRange.layerCount` must be non-zero and `subresourceRange.baseArrayLayer+subresourceRange.layerCount` must be less than or equal to the depth computed from `baseMipLevel` and `extent.depth` specified in `VkImageCreateInfo` when `image` was created, according to the formula defined in Image Miplevel Sizing.
18. 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
19. If `image` was created with the `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT` flag, but without the `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT` flag, and if the `format` of the `image` is not a `multi-planar` format, `format` must be compatible with the `format` used to create `image`, as defined in Format Compatibility Classes
20. If `image` was created with the `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT` flag, `format` must be compatible with, or must be an uncompressed format that is size-compatible with, the `format` used to create `image`.
21. If `image` was created with the `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT` flag, the `levelCount` and `layerCount` members of `subresourceRange` must both be `1`.
22. If a `VkImageFormatListCreateInfo` structure was included in the `pNext` chain of the `VkImageCreateInfo` structure used when creating `image` and the `viewFormatCount` field of `VkImageFormatListCreateInfo` is not zero then `format` must be one of the formats in

VkImageFormatListCreateInfo::pViewFormats.
23. If image was created with the VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT flag, if the format of the image is a multi-planar format, and if subresourceRange.aspectMask is one of VK_IMAGE_ASPECT_PLANE_0_BIT, VK_IMAGE_ASPECT_PLANE_1_BIT, or VK_IMAGE_ASPECT_PLANE_2_BIT, then format must be compatible with the VkFormat for the plane of the image format indicated by subresourceRange.aspectMask, as defined in Compatible formats of planes of multi-planar formats
24. If image was not created with the VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT flag, or if the format of the image is a multi-planar format and if subresourceRange.aspectMask is VK_IMAGE_ASPECT_COLOR_BIT, format must be identical to the format used to create image
25. If the pNext chain includes a VkSamplerYcbcrConversionInfo structure with a conversion value other than VK_NULL_HANDLE, all members of components must have the value VK_COMPONENT_SWIZZLE_IDENTITY.
26. If image is non-sparse then it must be bound completely and contiguously to a single VkDeviceMemory object
27. subresourceRange and viewType must be compatible with the image, as described in the compatibility table
28. If image has an external format, format must be VK_FORMAT_UNDEFINED.
29. If image has an external format, the pNext chain must include a VkSamplerYcbcrConversionInfo structure with a conversion object created with the same external format as image.
30. If image has an external format, all members of components must be VK_COMPONENT_SWIZZLE_IDENTITY.
31. If image was created with usage containing VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV, viewType must be VK_IMAGE_VIEW_TYPE_2D or VK_IMAGE_VIEW_TYPE_2D_ARRAY
32. If image was created with usage containing VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV, format must be VK_FORMAT_R8_UINT
33. If dynamic fragment density map feature is not enabled, flags must not contain VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DYNAMIC_BIT_EXT
34. If dynamic fragment density map feature is not enabled and image was created with usage containing VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT, flags must not contain any of VK_IMAGE_CREATE_PROTECTED_BIT, VK_IMAGE_CREATE_SPARSE_BINDING_BIT, VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT, or VK_IMAGE_CREATE_SPARSE_ALIASED_BIT
35. If the pNext chain includes a VkImageViewUsageCreateInfo structure, and image was not created with a VkImageStencilUsageCreateInfo structure included in the pNext chain of VkImageCreateInfo, its usage member must not include any bits that were not set in the usage member of the VkImageCreateInfo structure used to create image
36. If the pNext chain includes a VkImageViewUsageCreateInfo structure, image was created with a VkImageStencilUsageCreateInfo structure included in the pNext chain of VkImageCreateInfo, and subResourceRange.aspectMask includes VK_IMAGE_ASPECT_STENCIL_BIT, the usage member of the VkImageViewUsageCreateInfo instance must not include any bits that were not set in the usage member of the VkImageStencilUsageCreateInfo structure used to create image
37. If the pNext chain includes a VkImageViewUsageCreateInfo structure, image was created with a VkImageStencilUsageCreateInfo structure included in the pNext chain of VkImageCreateInfo, and subResourceRange.aspectMask includes bits other than VK_IMAGE_ASPECT_STENCIL_BIT, the usage member of the VkImageViewUsageCreateInfo structure must not include any bits that were not set in the usage member of the VkImageCreateInfo structure used to create image

Validations for VkImageSubresourceRange:

1. If levelCount is not VK_REMAINING_MIP_LEVELS, it must be greater than 0
2. If layerCount is not VK_REMAINING_ARRAY_LAYERS, it must be greater than 0
3. If aspectMask includes VK_IMAGE_ASPECT_COLOR_BIT, then it must not include any of

VK_IMAGE_ASPECT_PLANE_0_BIT, VK_IMAGE_ASPECT_PLANE_1_BIT, or VK_IMAGE_ASPECT_PLANE_2_BIT
4. aspectMask must not include VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT for any index i

Sempahore

Validations for VkSemaphoreTypeCreateInfo:

1. If the timelineSemaphore feature is not enabled, semaphoreType must not equal VK_SEMAPHORE_TYPE_TIMELINE
2. If semaphoreType is VK_SEMAPHORE_TYPE_BINARY, initialValue must be zero.
    - Handled by API design

Usage validations

Validations of correct unsage in other functions as dictated by the Vulkan specification.

Queue

Validations for vkGetDeviceQueue:

1. queueFamilyIndex must be one of the queue family indices specified when device was created, via the VkDeviceQueueCreateInfo structure
    - Handled by API design
2. 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
    - Handled by API design
3. VkDeviceQueueCreateInfo::flags must have been set to zero when device was created
    - Handled by API design

Validations for vkGetDeviceQueue2:

Validations for vkQueueSubmit:

1. If fence is not VK_NULL_HANDLE, fence must be unsignaled
2. 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
3. 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*
4. 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
5. Each 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
6. When a semaphore wait operation referring to a binary semaphore defined by any element of the pWaitSemaphores member of any element of pSubmits executes on queue, there must be no other queues waiting on the same semaphore
7. All elements of the pWaitSemaphores member of all elements of pSubmits created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_BINARY must reference a semaphore signal

operation that has been submitted for execution and any semaphore signal operations on which it depends (if any) must have also been submitted for execution
8. Each element of the `pCommandBuffers` member of each element of `pSubmits` must be in the pending or executable state
9. If any element of the `pCommandBuffers` member of any element of `pSubmits` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT`, it must not be in the pending state
10. Any secondary command buffers recorded into any element of the `pCommandBuffers` member of any element of `pSubmits` must be in the pending or executable state
11. If any secondary command buffers recorded into any element of the `pCommandBuffers` member of any element of `pSubmits` was not recorded with the `VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT`, it must not be in the pending state
12. Each element of the `pCommandBuffers` member of each element of `pSubmits` must have been allocated from a `VkCommandPool` that was created for the same queue family `queue` belongs to
    - Returns error
13. If any element of `pSubmits→pCommandBuffers` includes a Queue Family Transfer Acquire Operation, there must exist a previously submitted Queue Family Transfer Release Operation on a queue in the queue family identified by the acquire operation, with parameters matching the acquire operation as defined in the definition of such acquire operations, and which happens before the acquire operation
14. If a command recorded into any element of `pCommandBuffers` was a `vkCmdBeginQuery` whose `queryPool` was created with a `queryType` of `VK_QUERY_TYPE_PERFORMANCE_QUERY_KHR`, the profiling lock must have been held continuously on the `VkDevice` that `queue` was retrieved from, throughout recording of those command buffers
15. Any resource created with `VK_SHARING_MODE_EXCLUSIVE` that is read by an operation specified by `pSubmits` must not be owned by any queue family other than the one which `queue` belongs to, at the time it is executed

Validations for `VkSubmitInfo`:

1. Each element of `pCommandBuffers` must not have been allocated with `VK_COMMAND_BUFFER_LEVEL_SECONDARY`
2. If the geometry shaders feature is not enabled, each element of `pWaitDstStageMask` must not contain `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
3. If the tessellation shaders feature is not enabled, each element of `pWaitDstStageMask` must not contain `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT` or `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
4. Each element of `pWaitDstStageMask` must not include `VK_PIPELINE_STAGE_HOST_BIT`.
5. If any element of `pWaitSemaphores` or `pSignalSemaphores` was created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`, then the `pNext` chain must include a `VkTimelineSemaphoreSubmitInfo` structure
6. If the `pNext` chain of this structure includes a `VkTimelineSemaphoreSubmitInfo` structure and any element of `pWaitSemaphores` was created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`, then its `waitSemaphoreValueCount` member must equal `waitSemaphoreCount`
7. If the `pNext` chain of this structure includes a `VkTimelineSemaphoreSubmitInfo` structure and any element of `pSignalSemaphores` was created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE`, then its `signalSemaphoreValueCount` member must equal `signalSemaphoreCount`
8. For each element of `pSignalSemaphores` created with a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_TIMELINE` the corresponding element of `VkTimelineSemaphoreSubmitInfo`::pSignalSemaphoreValues must have a value greater than the current value of the semaphore when the semaphore signal operation is executed
9. For each element of `pWaitSemaphores` created with a `VkSemaphoreType` of

VK_SEMAPHORE_TYPE_TIMELINE the corresponding element of VkTimelineSemaphoreSubmitInfo::pWaitSemaphoreValues must have a value which does not differ from the current value of the semaphore or the value of any outstanding semaphore wait or signal operation on that semaphore by more than maxTimelineSemaphoreValueDifference.

10. For each element of pSignalSemaphores created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_TIMELINE the corresponding element of VkTimelineSemaphoreSubmitInfo::pSignalSemaphoreValues must have a value which does not differ from the current value of the semaphore or the value of any outstanding semaphore wait or signal operation on that semaphore by more than maxTimelineSemaphoreValueDifference.
11. If the mesh shaders feature is not enabled, each element of pWaitDstStageMask must not contain VK_PIPELINE_STAGE_MESH_SHADER_BIT_NV
12. If the task shaders feature is not enabled, each element of pWaitDstStageMask must not contain VK_PIPELINE_STAGE_TASK_SHADER_BIT_NV

Swapchain

Validations for vkAcquireNextImageKHR:

1. swapchain must not be in the retired state
2. If semaphore is not VK_NULL_HANDLE it must be unsignaled
3. If semaphore is not VK_NULL_HANDLE it must not have any uncompleted signal or wait operations pending
4. If fence is not VK_NULL_HANDLE it must be unsignaled and must not be associated with any other queue command that has not yet completed execution on that queue
5. semaphore and fence must not both be equal to VK_NULL_HANDLE
6. If the number of currently acquired images is greater than the difference between the number of images in swapchain and the value of VkSurfaceCapabilitiesKHR::minImageCount as returned by a call to vkGetPhysicalDeviceSurfaceCapabilities2KHR with the surface used to create swapchain, timeout must not be UINT64_MAX
7. semaphore must have a VkSemaphoreType of VK_SEMAPHORE_TYPE_BINARY

Validations for vkQueuePresentKHR:

1. Each element of pSwapchains member of pPresentInfo must be a swapchain that is created for a surface for which presentation is supported from queue as determined using a call to vkGetPhysicalDeviceSurfaceSupportKHR
2. If more than one member of pSwapchains was created from a display surface, all display surfaces referenced that refer to the same display must use the same display mode
3. When a semaphore wait operation referring to a binary semaphore defined by the elements of the pWaitSemaphores member of pPresentInfo executes on queue, there must be no other queues waiting on the same semaphore.
4. All elements of the pWaitSemaphores member of pPresentInfo must be semaphores that are signaled, or have semaphore signal operations previously submitted for execution.
5. All elements of the pWaitSemaphores member of pPresentInfo must be created with a VkSemaphoreType of VK_SEMAPHORE_TYPE_BINARY.
    - Handled by API design
6. All elements of the pWaitSemaphores member of pPresentInfo must reference a semaphore signal operation that has been submitted for execution and any semaphore signal operations on which it depends (if any) must have also been submitted for execution.

Validations for VkPresentInfoKHR:

1. Each element of `pImageIndices` must be the index of a presentable image acquired from the swapchain specified by the corresponding element of the `pSwapchains` array, and the presented image subresource must be in the `VK_IMAGE_LAYOUT_PRESENT_SRC_KHR` or `VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR` layout at the time the operation is executed on a `VkDevice`
   - Guaranteed by the type system
2. All elements of the `pWaitSemaphores` must have a `VkSemaphoreType` of `VK_SEMAPHORE_TYPE_BINARY`

Fence

Validations for `vkResetFences`:

1. Each element of `pFences` must not be currently associated with any queue command that has not yet completed execution on that queue

Statistics

Category	Statically solved	Dynamically solved	Left to user	Total
Implicit	148	15	0	163
Creation	43	0	132	175
Usage	5	1	40	46
Total	196	16	172	384