

Plan of implementation

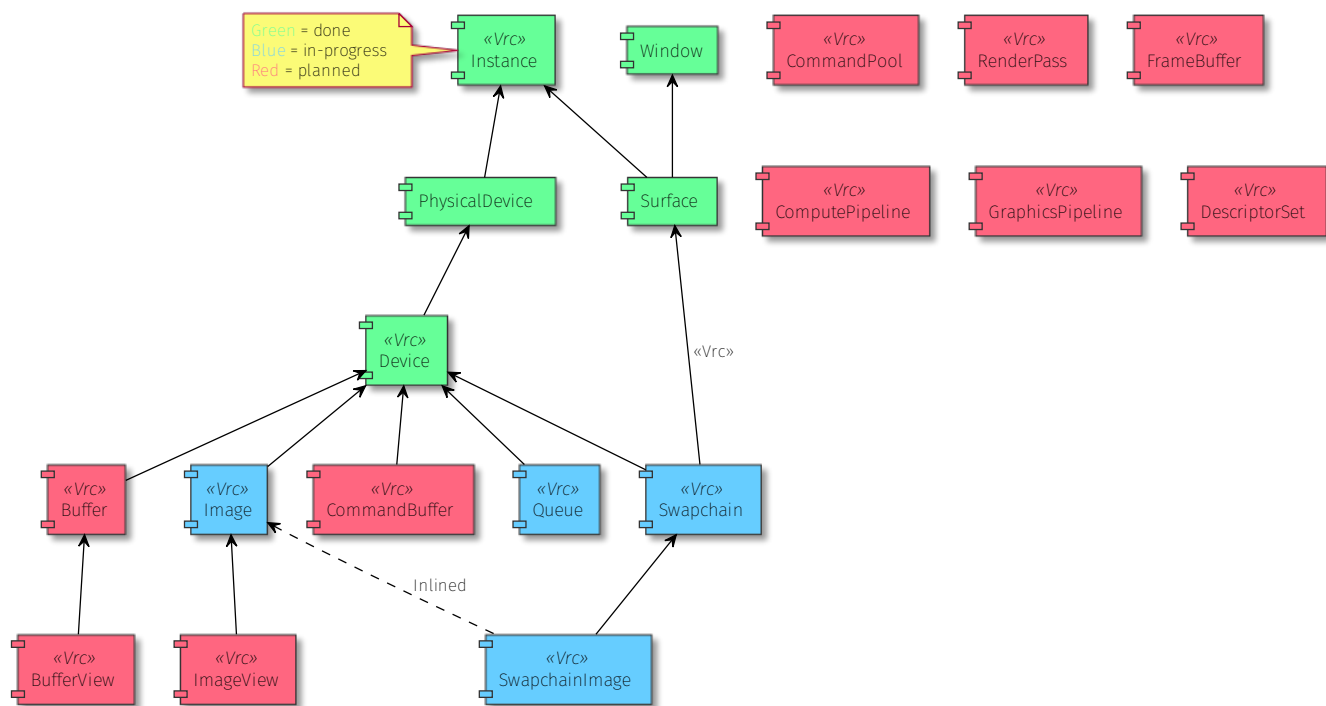


Figure 1: Object Dependency Graph

Validation

Validation of correct usage as dictated by the Vulkan specification.

Instance

Validations for `vkCreateInstance`:

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

Device

Validations for `vkCreateDevice`:

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

Validations for `VkDeviceCreateInfo`:

- 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
 - Left to validation layers
- If the `pNext` chain includes a `VkPhysicalDeviceFeatures2` structure, then `pEnabledFeatures` must be `NULL`
 - `pNext` chain is not exposed through the API
- `ppEnabledExtensionNames` must not contain `VK_AMD_negative_viewport_height`
- `ppEnabledExtensionNames` must not contain both `VK_KHR_buffer_device_address` and `VK_EXT_buffer_device_address`
- If the `pNext` chain includes a `VkPhysicalDeviceVulkan11Features` structure, then it must not include a `VkPhysicalDevice16BitStorageFeatures`, `VkPhysicalDeviceMultiviewFeatures`, `VkPhysicalDeviceVariablePointersFeatures`, `VkPhysicalDeviceProtectedMemoryFeatures`, `VkPhysicalDeviceSamplerYcbcrConversionFeatures`, or `VkPhysicalDeviceShaderDrawParametersFeatures` structure
 - `pNext` chain is not exposed through the API
- If the `pNext` chain includes a `VkPhysicalDeviceVulkan12Features` structure, then it must not include a `VkPhysicalDevice8BitStorageFeatures`, `VkPhysicalDeviceShaderAtomicInt64Features`, `VkPhysicalDeviceShaderFloat16Int8Features`, `VkPhysicalDeviceDescriptorIndexingFeatures`, `VkPhysicalDeviceScalarBlockLayoutFeatures`, `VkPhysicalDeviceImagelessFramebufferFeatures`, `VkPhysicalDeviceUniformBufferStandardLayoutFeatures`, `VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures`, `VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures`, `VkPhysicalDeviceHostQueryResetFeatures`, `VkPhysicalDeviceTimelineSemaphoreFeatures`, `VkPhysicalDeviceBufferDeviceAddressFeatures`, or `VkPhysicalDeviceVulkanMemoryModelFeatures` structure
 - `pNext` chain is not exposed through the API
- If `ppEnabledExtensions` contains code:“`VK_KHR_draw_indirect_count`” and the `pNext` chain includes a `VkPhysicalDeviceVulkan12Features` structure, then `VkPhysicalDeviceVulkan12Features::drawIndirectCount` must be `VK_TRUE`
 - `pNext` chain is not exposed through the API
- 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`
 - `pNext` chain is not exposed through the API
- If `ppEnabledExtensions` contains code:“`VK_EXT_descriptor_indexing`” and the `pNext` chain includes a `VkPhysicalDeviceVulkan12Features` structure, then `VkPhysicalDeviceVulkan12Features::descriptorIndexing` must be `VK_TRUE`
 - `pNext` chain is not exposed through the API
- If `ppEnabledExtensions` contains code:“`VK_EXT_sampler_filter_minmax`” and the `pNext` chain includes a `VkPhysicalDeviceVulkan12Features` structure, then `VkPhysicalDeviceVulkan12Features::samplerFilterMinmax` must be `VK_TRUE`
 - `pNext` chain is not exposed through the API
- 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`
 - `pNext` chain is not exposed through the API

Queue

Validations for `VkDeviceQueueCreateInfo`:

- `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
- If the protected memory feature is not enabled, the `VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT` bit of flags must not be set.
 - flags are not exposed through the API

Swapchain

Validations for `VkSwapchainCreateInfoKHR`:

- `surface` must be a surface that is supported by the device as determined using `vkGetPhysicalDeviceSurfaceSupportKHR`
- `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
- 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
- `minImageCount` must be 1 if `presentMode` is either `VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` or `VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR`
- `imageFormat` and `imageColorSpace` must match the `format` and `colorSpace` members, respectively, of one of the `VkSurfaceFormatKHR` structures returned by `vkGetPhysicalDeviceSurfaceFormatsKHR` for the surface
- `imageExtent` must be between `minImageExtent` and `maxImageExtent`, inclusive, where `minImageExtent` and `maxImageExtent` are members of the `VkSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` for the surface
- `imageExtent` members `width` and `height` must both be non-zero
 - Guaranteed by the type system
- `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
- 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
- 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

- 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
- If `imageSharingMode` is `VK_SHARING_MODE_CONCURRENT`, `queueFamilyIndexCount` must be greater than 1
 - Guaranteed by the type system
- 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
- `preTransform` must be one of the bits present in the `supportedTransforms` member of the `VkSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` for the surface
- `compositeAlpha` must be one of the bits present in the `supportedCompositeAlpha` member of the `VkSurfaceCapabilitiesKHR` structure returned by `vkGetPhysicalDeviceSurfaceCapabilitiesKHR` for the surface
- `presentMode` must be one of the `VkPresentModeKHR` values returned by `vkGetPhysicalDeviceSurfacePresentModesKHR` for the surface
- 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`
 - flags are not exposed through the API
- If `oldSwapchain` is not `VK_NULL_HANDLE`, `oldSwapchain` must be a non-retired swapchain associated with native window referred to by surface
 - Recreation is handled specially
- The implied image creation parameters of the swapchain must be supported as reported by `vkGetPhysicalDeviceImageFormatProperties`
- 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`
 - flags are not exposed through the API
- 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
 - flags are not exposed through the API
- 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
 - pNext chain is not exposed through the API

CommandPool

Validations for `vkCreateCommandPool`:

- `pCreateInfo→queueFamilyIndex` must be the index of a queue family available in the logical device device.

Validations for `VkCommandPoolCreateInfo`:

- If the protected memory feature is not enabled, the `VK_COMMAND_POOL_CREATE_PROTECTED_BIT` bit of `flags` must not be set.

CommandBuffer

Validations for `VkCommandBufferAllocateInfo`:

- `commandBufferCount` must be greater than 0

RenderPass

Validations for `VkRenderPassCreateInfo2`:

- 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 `VkAttachmentDescription2` 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`
- For any member of `pAttachments` with a `loadOp` equal to `VK_ATTACHMENT_LOAD_OP_CLEAR`, the first use of that attachment must not specify a `layout` equal to `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`
- For any member of `pAttachments` with a `stencilLoadOp` equal to `VK_ATTACHMENT_LOAD_OP_CLEAR`, the first use of that attachment must not specify a `layout` equal to `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, or `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`.
- 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 destination subpass
- The set of bits included in any element of `pCorrelatedViewMasks` must not overlap with the set of bits included in any other element of `pCorrelatedViewMasks`
- If the `VkSubpassDescription2::viewMask` member of all elements of `pSubpasses` is 0, `correlatedViewMaskCount` must be 0
- The `VkSubpassDescription2::viewMask` member of all elements of `pSubpasses` must either all be 0, or all not be 0
- If the `VkSubpassDescription2::viewMask` member of all elements of `pSubpasses` is 0, the `dependencyFlags` member of any element of `pDependencies` must not include `VK_DEPENDENCY_VIEW_LOCAL_BIT`

- For any element of `pDependencies` where its `srcSubpass` member equals its `dstSubpass` member, if the `viewMask` member of the corresponding element of `pSubpasses` includes more than one bit, its `dependencyFlags` member must include `VK_DEPENDENCY_VIEW_LOCAL_BIT`
- The `viewMask` member must not have a bit set at an index greater than or equal to `VkPhysicalDeviceLimits::maxFramebufferLayers`
- If the `attachment` member of any element of the `pInputAttachments` member of any element of `pSubpasses` is not `VK_ATTACHMENT_UNUSED`, the `aspectMask` member of that element of `pInputAttachments` must only include aspects that are present in images of the format specified by the element of `pAttachments` specified by `attachment`
- The `srcSubpass` member of each element of `pDependencies` must be less than `subpassCount`
- The `dstSubpass` member of each element of `pDependencies` must be less than `subpassCount`

Validations for `VkAttachmentDescription2`:

- `finalLayout` must not be `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`
- If `format` is a color format, `initialLayout` must not be `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`
- If `format` is a depth/stencil format, `initialLayout` must not be `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`
- If `format` is a color format, `finalLayout` must not be `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL`, or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL`
- If `format` is a depth/stencil format, `finalLayout` must not be `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`
- If the `separateDepthStencilLayouts` feature is not enabled, `initialLayout` must not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- If the `separateDepthStencilLayouts` feature is not enabled, `finalLayout` must not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- If `format` is a color format, `initialLayout` must not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- If `format` is a color format, `finalLayout` must not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL`, `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- If `format` is a depth/stencil format which includes both depth and stencil aspects, and `initialLayout` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, the `pNext` chain must include a `VkAttachmentDescriptionStencilLayout` structure
- If `format` is a depth/stencil format which includes both depth and stencil aspects, and

`finalLayout` is `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`, the `pNext` chain must include a `VkAttachmentDescriptionStencilLayout` structure

- If `format` is a depth/stencil format which includes only the depth aspect, `initialLayout` must not be `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- If `format` is a depth/stencil format which includes only the depth aspect, `finalLayout` must not be `VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL`
- If `format` is a depth/stencil format which includes only the stencil aspect, `initialLayout` must not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`
- If `format` is a depth/stencil format which includes only the stencil aspect, `finalLayout` must not be `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL`

Validations for `VkSubpassDescription2`:

- `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`, for each resolve attachment that is not `VK_ATTACHMENT_UNUSED`, the corresponding color attachment must not have a sample count of `VK_SAMPLE_COUNT_1_BIT`
- If `pResolveAttachments` is not `NULL`, each resolve attachment that is not `VK_ATTACHMENT_UNUSED` must have a sample count of `VK_SAMPLE_COUNT_1_BIT`
- 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 the `VK_AMD_mixed_attachment_samples` extension is enabled, all attachments in `pColorAttachments` that are not `VK_ATTACHMENT_UNUSED` must have a sample count that is smaller than or equal to the sample count of `pDepthStencilAttachment` if it is not `VK_ATTACHMENT_UNUSED`
- If neither the `VK_AMD_mixed_attachment_samples` nor the `VK_NV_framebuffer_mixed_samples` extensions are enabled, and if `pDepthStencilAttachment` is not `VK_ATTACHMENT_UNUSED` and any attachments in `pColorAttachments` are not `VK_ATTACHMENT_UNUSED`, they must have the same sample count
- 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 by more than one `VkAttachmentReference` member, then each use must use the same layout
- If `flags` includes `VK_SUBPASS_DESCRIPTION_PER_VIEW_POSITION_X_ONLY_BIT_NVX`, it must also include `VK_SUBPASS_DESCRIPTION_PER_VIEW_ATTRIBUTES_BIT_NVX`.
- If the attachment member of any element of `pInputAttachments` is not `VK_ATTACHMENT_UNUSED`, then the `aspectMask` member must be a valid combination of `VkImageAspectFlagBits`

- If the attachment member of any element of pInputAttachments is not VK_ATTACHMENT_UNUSED, then the aspectMask member must not be 0
- If the attachment member of any element of pInputAttachments is not VK_ATTACHMENT_UNUSED, then the aspectMask member must not include VK_IMAGE_ASPECT_METADATA_BIT

Validations for VkAttachmentReference2:

- If attachment is not VK_ATTACHMENT_UNUSED, layout must not be VK_IMAGE_LAYOUT_UNDEFINED, VK_IMAGE_LAYOUT_PREINITIALIZED, or VK_IMAGE_LAYOUT_PRESENT_SRC_KHR
- If attachment is not VK_ATTACHMENT_UNUSED, and aspectMask does not include VK_IMAGE_ASPECT_STENCIL_BIT or VK_IMAGE_ASPECT_DEPTH_BIT, layout must not be VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL, or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL
- If attachment is not VK_ATTACHMENT_UNUSED, and aspectMask does not include VK_IMAGE_ASPECT_COLOR_BIT, layout must not be VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
- If the separateDepthStencilLayouts feature is not enabled, and attachment is not VK_ATTACHMENT_UNUSED, layout must not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- If attachment is not VK_ATTACHMENT_UNUSED, and aspectMask includes VK_IMAGE_ASPECT_COLOR_BIT, layout must not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- If attachment is not VK_ATTACHMENT_UNUSED, and aspectMask includes both VK_IMAGE_ASPECT_DEPTH_BIT and VK_IMAGE_ASPECT_STENCIL_BIT, and layout is VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL, the pNext chain must include a VkAttachmentReferenceStencilLayout structure
- If attachment is not VK_ATTACHMENT_UNUSED, and aspectMask includes only VK_IMAGE_ASPECT_DEPTH_BIT then layout must not be VK_IMAGE_LAYOUT_STENCIL_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_STENCIL_READ_ONLY_OPTIMAL
- If attachment is not VK_ATTACHMENT_UNUSED, and aspectMask includes only VK_IMAGE_ASPECT_STENCIL_BIT then layout must not be VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, or VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_OPTIMAL

Validations for VkSubpassDependency2:

- 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` 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
- If `dependencyFlags` includes `VK_DEPENDENCY_VIEW_LOCAL_BIT`, `srcSubpass` must not be equal to `VK_SUBPASS_EXTERNAL`
- If `dependencyFlags` includes `VK_DEPENDENCY_VIEW_LOCAL_BIT`, `dstSubpass` must not be equal to `VK_SUBPASS_EXTERNAL`
- If `srcSubpass` equals `dstSubpass`, and `srcStageMask` and `dstStageMask` both include a framebuffer-space stage, then `dependencyFlags` must include `VK_DEPENDENCY_BY_REGION_BIT`
- If `viewOffset` is not equal to 0, `srcSubpass` must not be equal to `dstSubpass`
- If `dependencyFlags` does not include `VK_DEPENDENCY_VIEW_LOCAL_BIT`, `viewOffset` must be 0
- If `viewOffset` is not 0, `srcSubpass` must not be equal to `dstSubpass`.
- If the mesh shaders feature is not enabled, `srcStageMask` must not contain `VK_PIPELINE_STAGE_MESH_SHADER_BIT_NV`
- If the task shaders feature is not enabled, `srcStageMask` must not contain `VK_PIPELINE_STAGE_TASK_SHADER_BIT_NV`
- If the mesh shaders feature is not enabled, `dstStageMask` must not contain `VK_PIPELINE_STAGE_MESH_SHADER_BIT_NV`
- If the task shaders feature is not enabled, `dstStageMask` must not contain `VK_PIPELINE_STAGE_TASK_SHADER_BIT_NV`

Framebuffer

Validations for `vkCreateFramebuffer`:

- If `pCreateInfo->flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, and `attachmentCount` is not 0, each element of `pCreateInfo->pAttachments` must have been created on device

Validations for `VkFramebufferCreateInfo`:

- `attachmentCount` must be equal to the attachment count specified in `renderPass`
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, and `attachmentCount` is not 0, `pAttachments` must be a valid pointer to an array of `attachmentCount` valid `VkImageView` handles
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as a color attachment or resolve attachment by `renderPass` must have been created with a `usage` value including `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as a depth/stencil attachment by `renderPass` must have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of

`pAttachments` that is used as a depth/stencil resolve attachment by `renderPass` must have been created with a `usage` value including `VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT`

- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is used as an input attachment by `renderPass` must have been created with a `usage` value including `VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT`
- Each element of `pAttachments` that is used as a fragment density map attachment by `renderPass` must not have been created with a `flags` value including `VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT`.
- If `renderPass` has a fragment density map attachment and non-subsample image feature is not enabled, each element of `pAttachments` must have been created with a `flags` value including `VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT` unless that element is the fragment density map attachment.
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` must have been created with a `VkFormat` value that matches the `VkFormat` specified by the corresponding `VkAttachmentDescription` in `renderPass`
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` must have been created with a `samples` value that matches the `samples` value specified by the corresponding `VkAttachmentDescription` in `renderPass`
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` must have dimensions at least as large as the corresponding framebuffer dimension except for any element that is referenced by `fragmentDensityMapAttachment`
- If `renderPass` was specified with non-zero view masks, each element of `pAttachments` that is not referenced by `fragmentDensityMapAttachment` must have a `layerCount` greater than the index of the most significant bit set in any of those view masks
- If `renderPass` was specified with non-zero view masks, each element of `pAttachments` that is referenced by `fragmentDensityMapAttachment` must have a `layerCount` equal to 1 or greater than the index of the most significant bit set in any of those view masks
- If `renderPass` was not specified with non-zero view masks, each element of `pAttachments` that is referenced by `fragmentDensityMapAttachment` must have a `layerCount` equal to 1
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, an element of `pAttachments` that is referenced by `fragmentDensityMapAttachment` must have a width at least as large as `ceil(width / maxFragmentDensityTexelSize_width)`
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, an element of `pAttachments` that is referenced by `fragmentDensityMapAttachment` must have a height at least as large as `ceil(height / maxFragmentDensityTexelSize_height)`
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` must only specify a single mip level
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each 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`
- If `renderPass` was specified with non-zero view masks, layers must be 1
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, each element of `pAttachments` that is a 2D or 2D array image view taken from a 3D image must not be a depth/stencil format
- If `flags` does not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, and `attachmentCount` is not 0, `pAttachments` must be a valid pointer to an array of `attachmentCount` valid `VkImageView` handles
- If the imageless framebuffer feature is not enabled, `flags` must not include `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`
- If `flags` includes `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`, the `pNext` chain must include a

VkFramebufferAttachmentsCreateInfo structure

- If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the attachmentImageInfoCount member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain must be equal to either zero or attachmentCount
- If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the width member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain must be greater than or equal to width, except for any element that is referenced by
VkRenderPassFragmentDensityMapCreateInfoEXT::fragmentDensityMapAttachment in renderPass
- If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the height member of any element of the pAttachmentImageInfos member of a
VkFramebufferAttachmentsCreateInfo structure included in the pNext chain must be greater than or equal to height, except for any element that is referenced by
VkRenderPassFragmentDensityMapCreateInfoEXT::fragmentDensityMapAttachment in renderPass
- If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the width member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain that is referenced by
VkRenderPassFragmentDensityMapCreateInfoEXT::fragmentDensityMapAttachment in renderPass must be greater than or equal to $\text{ceil}(\text{width} / \text{maxFragmentDensityTexelSize_width})$
- If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the height member of any element of the pAttachmentImageInfos member of a
VkFramebufferAttachmentsCreateInfo structure included in the pNext chain that is referenced by
VkRenderPassFragmentDensityMapCreateInfoEXT::fragmentDensityMapAttachment in renderPass must be greater than or equal to $\text{ceil}(\text{height} / \text{maxFragmentDensityTexelSize_height})$
- If multiview is enabled for renderPass, and flags includes
VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the layerCount member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain must be greater than the maximum bit index set in the view mask in the subpasses in which it is used in renderPass
- If multiview is not enabled for renderPass, and flags includes
VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the layerCount member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain must be greater than or equal to layers
- If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the usage member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain that refers to an attachment used as a color attachment or resolve attachment by renderPass must include VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
- If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the usage member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain that refers to an attachment used as a depth/stencil attachment by renderPass must include VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the usage member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain that refers to an attachment used as a depth/stencil resolve attachment by renderPass must include VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
- If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, the usage member of any element of the pAttachmentImageInfos member of a VkFramebufferAttachmentsCreateInfo structure included in the pNext chain that refers to an attachment used as an input attachment by renderPass must include VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
- If flags includes VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT, at least one element of the

`pViewFormats` member of any element of the `pAttachmentImageInfos` member of a `VkFramebufferAttachmentsCreateInfo` structure included in the `pNext` chain must be equal to the corresponding value of `VkAttachmentDescription::format` used to create `renderPass`

ComputePipeline

Validations for `vkCreateComputePipelines`:

- If the `flags` member of any 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 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

Validations for `VkComputePipelineCreateInfo`:

- 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`
- The number of resources in `layout` accessible to the compute shader stage must be less than or equal to `VkPhysicalDeviceLimits::maxPerStageResources`

Validations for `VkPipelineShaderStageCreateInfo`:

- 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`
- If the mesh shader feature is not enabled, `stage` must not be `VK_SHADER_STAGE_MESH_BIT_NV`
- If the task shader feature is not enabled, `stage` must not be `VK_SHADER_STAGE_TASK_BIT_NV`
- `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 a vertex processing stage, 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 a vertex processing stage, 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
- If stage is `VK_SHADER_STAGE_FRAGMENT_BIT`, and the identified entry point writes to `FragStencilRefEXT` in any execution path, it must write to `FragStencilRefEXT` in all execution paths
- If stage is `VK_SHADER_STAGE_MESH_BIT_NV`, the identified entry point must have an `OpExecutionMode` instruction that specifies a maximum output vertex count, `OutputVertices`, that is greater than 0 and less than or equal to `VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshOutputVertices`.
- If stage is `VK_SHADER_STAGE_MESH_BIT_NV`, the identified entry point must have an `OpExecutionMode` instruction that specifies a maximum output primitive count, `OutputPrimitivesNV`, that is greater than 0 and less than or equal to `VkPhysicalDeviceMeshShaderPropertiesNV::maxMeshOutputPrimitives`.
- If `flags` has the `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT` flag set, the `subgroupSizeControl` feature must be enabled.
- If `flags` has the `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT` flag set, the `computeFullSubgroups` feature must be enabled.
- If a `VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT` structure is included in the `pNext` chain, `flags` must not have the `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT` flag set.
- If a `VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT` structure is included in the `pNext` chain, the `subgroupSizeControl` feature must be enabled, and `stage` must be a valid bit specified in `requiredSubgroupSizeStages`.
- If a `VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT` structure is included in the `pNext` chain and `stage` is `VK_SHADER_STAGE_COMPUTE_BIT`, the local workgroup size of the shader must be less than or equal to the product of `VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT::requiredSubgroupSize` and `maxComputeWorkgroupSubgroups`.
- If a `VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT` structure is included in the `pNext` chain, and `flags` has the

VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT flag set, the local workgroup size in the X dimension of the pipeline must be a multiple of `VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT::requiredSubgroupSize`.

- If `flags` has both the `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT` and `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT` flags set, the local workgroup size in the X dimension of the pipeline must be a multiple of `maxSubgroupSize`.
- If `flags` has the `VK_PIPELINE_SHADER_STAGE_CREATE_REQUIRE_FULL_SUBGROUPS_BIT_EXT` flag set and `flags` does not have the `VK_PIPELINE_SHADER_STAGE_CREATE_ALLOW_VARYING_SUBGROUP_SIZE_BIT_EXT` flag set and no `VkPipelineShaderStageRequiredSubgroupSizeCreateInfoEXT` structure is included in the `pNext` chain, the local workgroup size in the X dimension of the pipeline must be a multiple of `subgroupSize`.

GraphicsPipeline

Validations for `vkCreateGraphicsPipelines`:

- If the `flags` member of any 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 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

Validations for `VkGraphicsPipelineCreateInfo`:

- 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 geometric shader stages provided in `pStages` must be either from the mesh shading pipeline (`stage` is `VK_SHADER_STAGE_TASK_BIT_NV` or `VK_SHADER_STAGE_MESH_BIT_NV`) or from the primitive shading pipeline (`stage` is `VK_SHADER_STAGE_VERTEX_BIT`, `VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT`, `VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT`, or `VK_SHADER_STAGE_GEOMETRY_BIT`).
- The `stage` member of one element of `pStages` must be either `VK_SHADER_STAGE_VERTEX_BIT` or `VK_SHADER_STAGE_MESH_BIT_NV`.
- The `stage` member of each 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 valid 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` or `VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_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` or `VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_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 attached image's format features does not contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT`.
- 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 valid pointer to an array of `pViewportState→viewportCount` valid `VkViewport` structures
- If no element of the `pDynamicStates` member of `pDynamicState` is `VK_DYNAMIC_STATE_SCISSOR`, the `pScissors` member of `pViewportState` must be a valid 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 valid pointer to a valid `VkPipelineViewportStateCreateInfo` structure
- If the `rasterizerDiscardEnable` member of `pRasterizationState` is `VK_FALSE`, `pMultisampleState` must be a valid 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 valid pointer to a valid `VkPipelineDepthStencilStateCreateInfo` structure
- If the `rasterizerDiscardEnable` member of `pRasterizationState` is `VK_FALSE`, and subpass uses color attachments, `pColorBlendState` must be a valid 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 `pRasterizationState` is `VK_TRUE`, the `depthBiasClamp` member of `pRasterizationState` must be `0.0`
- If the `VK_EXT_depth_range_unrestricted` extension is not enabled and no element of the `pDynamicStates` member of `pDynamicState` is `VK_DYNAMIC_STATE_DEPTH_BOUNDS`, and the `depthBoundsTestEnable` member of `pDepthStencilState` is `VK_TRUE`, the `minDepthBounds` and `maxDepthBounds` members of `pDepthStencilState` must be between `0.0` and `1.0`, inclusive
- If no element of the `pDynamicStates` member of `pDynamicState` is `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT`, and the `sampleLocationsEnable` member of a `VkPipelineSampleLocationsStateCreateInfoEXT` structure included in the `pNext` chain of `pMultisampleState` is `VK_TRUE`, `sampleLocationsInfo.sampleLocationGridSize.width` must evenly divide `VkMultisamplePropertiesEXT::sampleLocationGridSize.width` as returned by `vkGetPhysicalDeviceMultisamplePropertiesEXT` with a `samples` parameter equaling `rasterizationSamples`
- If no element of the `pDynamicStates` member of `pDynamicState` is `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT`, and the `sampleLocationsEnable` member of a `VkPipelineSampleLocationsStateCreateInfoEXT` structure included in the `pNext` chain of `pMultisampleState` is `VK_TRUE`, `sampleLocationsInfo.sampleLocationGridSize.height` must evenly divide `VkMultisamplePropertiesEXT::sampleLocationGridSize.height` as returned by `vkGetPhysicalDeviceMultisamplePropertiesEXT` with a `samples` parameter equaling `rasterizationSamples`
- If no element of the `pDynamicStates` member of `pDynamicState` is `VK_DYNAMIC_STATE_SAMPLE_LOCATIONS_EXT`, and the `sampleLocationsEnable` member of a `VkPipelineSampleLocationsStateCreateInfoEXT` structure included in the `pNext` chain of `pMultisampleState` is `VK_TRUE`, `sampleLocationsInfo.sampleLocationsPerPixel` must equal `rasterizationSamples`
- If the `sampleLocationsEnable` member of a `VkPipelineSampleLocationsStateCreateInfoEXT` structure included in the `pNext` chain of `pMultisampleState` is `VK_TRUE`, the fragment shader code must not statically use the extended instruction `InterpolateAtSample`
- layout must be consistent with all shaders specified in `pStages`
- If neither the `VK_AMD_mixed_attachment_samples` nor the `VK_NV_framebuffer_mixed_samples` extensions are enabled, and 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 the `VK_AMD_mixed_attachment_samples` extension is enabled, and if subpass uses color and/or depth/stencil attachments, then the `rasterizationSamples` member of `pMultisampleState` must equal the maximum of the sample counts of those subpass attachments
- If the `VK_NV_framebuffer_mixed_samples` extension is enabled, and if subpass has a depth/stencil attachment and depth test, stencil test, or depth bounds test are enabled, then the

`rasterizationSamples` member of `pMultisampleState` must be the same as the sample count of the depth/stencil attachment

- If the `VK_NV_framebuffer_mixed_samples` extension is enabled, and if `subpass` has any color attachments, then the `rasterizationSamples` member of `pMultisampleState` must be greater than or equal to the sample count for those subpass attachments
- If the `VK_NV_coverage_reduction_mode` extension is enabled, the coverage reduction mode specified by `VkPipelineCoverageReductionStateCreateInfoNV::coverageReductionMode`, the `rasterizationSamples` member of `pMultisampleState` and the sample counts for the color and depth/stencil attachments (if the subpass has them) must be a valid combination returned by `vkGetPhysicalDeviceSupportedFramebufferMixedSamplesCombinationsNV`
- 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`
- If the `renderPass` has multiview enabled and `subpass` has more than one bit set in the view mask and `multiviewTessellationShader` is not enabled, then `pStages` must not include tessellation shaders.
- If the `renderPass` has multiview enabled and `subpass` has more than one bit set in the view mask and `multiviewGeometryShader` is not enabled, then `pStages` must not include a geometry shader.
- If the `renderPass` has multiview enabled and `subpass` has more than one bit set in the view mask, shaders in the pipeline must not write to the `Layer` built-in output
- If the `renderPass` has multiview enabled, then all shaders must not include variables decorated with the `Layer` built-in decoration in their interfaces.
- `flags` must not contain the `VK_PIPELINE_CREATE_DISPATCH_BASE` flag.
- If `pStages` includes a fragment shader stage and an input attachment was referenced by the `VkRenderPassInputAttachmentAspectCreateInfo` at `renderPass` create time, its shader code must not read from any aspect that was not specified in the `aspectMask` of the corresponding `VkInputAttachmentAspectReference` structure.
- The number of resources in `layout` accessible to each shader stage that is used by the pipeline must be less than or equal to `VkPhysicalDeviceLimits::maxPerStageResources`
- If no element of the `pDynamicStates` member of `pDynamicState` is `VK_DYNAMIC_STATE_VIEWPORT_W_SCALING_NV`, and the `viewportWScalingEnable` member of a `VkPipelineViewportWScalingStateCreateInfoNV` structure, included in the `pNext` chain of `pViewportState`, is `VK_TRUE`, the `pViewportWScalings` member of the `VkPipelineViewportWScalingStateCreateInfoNV` must be a pointer to an array of `VkPipelineViewportWScalingStateCreateInfoNV::viewportCount` valid `VkViewportWScalingNV` structures
- If `pStages` includes a vertex shader stage, `pVertexInputState` must be a valid pointer to a valid `VkPipelineVertexInputStateCreateInfo` structure
- If `pStages` includes a vertex shader stage, `pInputAssemblyState` must be a valid pointer to a valid `VkPipelineInputAssemblyStateCreateInfo` structure
- The Xfb execution mode can be specified by only one shader stage in `pStages`
- If any shader stage in `pStages` specifies Xfb execution mode it must be the last vertex processing stage
- If a `VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream` value other than zero is specified, all variables in the output interface of the entry point being compiled decorated with `Position`, `PointSize`, `ClipDistance`, or `CullDistance` must all be decorated with identical `Stream` values that match the `rasterizationStream`
- If `VkPipelineRasterizationStateStreamCreateInfoEXT::rasterizationStream` is zero, or not specified, all variables in the output interface of the entry point being compiled decorated with `Position`, `PointSize`, `ClipDistance`, or `CullDistance` must all be decorated with a `Stream` value of zero, or must not specify the `Stream` decoration
- If the last vertex processing stage is a geometry shader, and that geometry shader uses the `GeometryStreams` capability, then

VkPhysicalDeviceTransformFeedbackFeaturesEXT::geometryStreams feature must be enabled

- If there are any mesh shader stages in the pipeline there must not be any shader stage in the pipeline with a Xfb execution mode.
- If the lineRasterizationMode member of a VkPipelineRasterizationLineStateCreateInfoEXT structure included in the pNext chain of pRasterizationState is VK_LINE_RASTERIZATION_MODE_BRESENHAM_EXT or VK_LINE_RASTERIZATION_MODE_RECTANGULAR_SMOOTH_EXT and if rasterization is enabled, then the alphaToCoverageEnable, alphaToOneEnable, and sampleShadingEnable members of pMultisampleState must all be VK_FALSE
- If the stippledLineEnable member of VkPipelineRasterizationLineStateCreateInfoEXT is VK_TRUE and no element of the pDynamicStates member of pDynamicState is VK_DYNAMIC_STATE_LINE_STIPPLE_EXT, then the lineStippleFactor member of VkPipelineRasterizationLineStateCreateInfoEXT must be in the range [1,256]

Buffer

Validations for vkCreateBuffer:

- 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

Validations for VkBufferCreateInfo:

- size must be greater than 0
- If sharingMode is VK_SHARING_MODE_CONCURRENT, pQueueFamilyIndices must be a valid 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 either vkGetPhysicalDeviceQueueFamilyProperties or vkGetPhysicalDeviceQueueFamilyProperties2 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
- If the pNext chain includes a VkExternalMemoryBufferCreateInfo structure, its handleTypes member must only contain bits that are also in VkExternalBufferProperties::externalMemoryProperties.compatibleHandleTypes, as returned by vkGetPhysicalDeviceExternalBufferProperties with pExternalBufferInfo->handleType equal to any one of the handle types specified in VkExternalMemoryBufferCreateInfo::handleTypes
- If the protected memory feature is not enabled, flags must not contain VK_BUFFER_CREATE_PROTECTED_BIT

- If any of the bits `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`, `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT`, or `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT` are set, `VK_BUFFER_CREATE_PROTECTED_BIT` must not also be set
- If the `pNext` chain includes a `VkDedicatedAllocationBufferCreateInfoNV` structure, and the `dedicatedAllocation` member of the chained structure is `VK_TRUE`, then `flags` must not include `VK_BUFFER_CREATE_SPARSE_BINDING_BIT`, `VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT`, or `VK_BUFFER_CREATE_SPARSE_ALIASED_BIT`
- If `VkBufferDeviceAddressCreateInfoEXT::deviceAddress` is not zero, `flags` must include `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`
- If `VkBufferOpaqueCaptureAddressCreateInfo::opaqueCaptureAddress` is not zero, `flags` must include `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`
- If `flags` includes `VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT`, the `bufferDeviceAddressCaptureReplay` or `VkPhysicalDeviceBufferDeviceAddressFeaturesEXT::bufferDeviceAddressCaptureReplay` feature must be enabled

BufferView

Validations for `VkBufferViewCreateInfo`:

- `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 an integer multiple of the texel block size of `format`
- If `range` is not equal to `VK_WHOLE_SIZE`, `range` divided by the texel block size of `format`, multiplied by the number of texels per texel block for that format (as defined in the Compatible Formats table), 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
- If the `texelBufferAlignment` feature is not enabled, `offset` must be a multiple of `VkPhysicalDeviceLimits::minTexelBufferOffsetAlignment`
- If the `texelBufferAlignment` feature is enabled and if `buffer` was created with `usage` containing `VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT`, `offset` must be a multiple of the lesser of `VkPhysicalDeviceTexelBufferAlignmentPropertiesEXT::storageTexelBufferOffsetAlignmentBytes` or, if `VkPhysicalDeviceTexelBufferAlignmentPropertiesEXT::storageTexelBufferOffsetSingleTexelAlignment` is `VK_TRUE`, the size of a texel of the requested `format`. If the size of a texel is a multiple of three bytes, then the size of a single

component of format is used instead

- If the `texelBufferAlignment` feature is enabled and if `buffer` was created with `usage` containing `VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT`, `offset` must be a multiple of the lesser of `VkPhysicalDeviceTexelBufferAlignmentPropertiesEXT::uniformTexelBufferOffsetAlignmentBytes` or, if `VkPhysicalDeviceTexelBufferAlignmentPropertiesEXT::uniformTexelBufferOffsetSingleTexelAlignment` is `VK_TRUE`, the size of a texel of the requested format. If the size of a texel is a multiple of three bytes, then the size of a single component of format is used instead

Image

Validations for `vkCreateImage`:

- 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`:

- Each of the following values (as described in Image Creation Limits) must not be undefined `imageCreateMaxMipLevels`, `imageCreateMaxArrayLayers`, `imageCreateMaxExtent`, and `imageCreateSampleCounts`.
- If `sharingMode` is `VK_SHARING_MODE_CONCURRENT`, `pQueueFamilyIndices` must be a valid 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 either `vkGetPhysicalDeviceQueueFamilyProperties` or `vkGetPhysicalDeviceQueueFamilyProperties2` for the `physicalDevice` that was used to create device
- If the `pNext` chain includes a `VkExternalFormatANDROID` structure, and its `externalFormat` member is non-zero the format must be `VK_FORMAT_UNDEFINED`.
- 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`.
- `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 `flags` contains `VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT`, `imageType` must be `VK_IMAGE_TYPE_2D`
- If `flags` contains `VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT`, `imageType` must be `VK_IMAGE_TYPE_3D`
- `extent.width` must be less than or equal to `imageCreateMaxExtent.width` (as defined in Image Creation Limits).
- `extent.height` must be less than or equal to `imageCreateMaxExtent.height` (as defined in Image Creation Limits).
- `extent.depth` must be less than or equal to `imageCreateMaxExtent.depth` (as defined in

Image Creation Limits).

- 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_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 the number of levels in the complete mipmap chain based on `extent.width`, `extent.height`, and `extent.depth`.
- `mipLevels` must be less than or equal to `imageCreateMaxMipLevels` (as defined in Image Creation Limits).
- `arrayLayers` must be less than or equal to `imageCreateMaxArrayLayers` (as defined in Image Creation Limits).
- If `imageType` is `VK_IMAGE_TYPE_3D`, `arrayLayers` must be 1.
- 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`,
- If `samples` is not `VK_SAMPLE_COUNT_1_BIT`, `usage` must not contain `VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT`
- 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_FRAGMENT_DENSITY_MAP_BIT_EXT`, `extent.width` must be less than or equal to `ceil(maxFramebufferWidth / minFragmentDensityTexelSize_width)`
- If `usage` includes `VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT`, `extent.height` must be less than or equal to `ceil(maxFramebufferHeight / minFragmentDensityTexelSize_height)`
- 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 `imageCreateSampleCounts` (as defined in Image Creation Limits).
- 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 the sparse aliased residency feature is not enabled, `flags` must not contain `VK_IMAGE_CREATE_SPARSE_ALIASED_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 `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`
- 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
- If the protected memory feature is not enabled, `flags` must not contain `VK_IMAGE_CREATE_PROTECTED_BIT`.
- 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.
- If the `pNext` chain includes a `VkExternalMemoryImageCreateInfoNV` structure, it must not contain a `VkExternalMemoryImageCreateInfo` structure.
- 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`
- 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`
- If the logical device was created with `VkDeviceGroupDeviceCreateInfo::physicalDeviceCount` equal to 1, `flags` must not contain `VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT`
- 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.
- 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.
- If `flags` contains `VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT`, then `flags` must also contain `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`.
- `initialLayout` must be `VK_IMAGE_LAYOUT_UNDEFINED` or `VK_IMAGE_LAYOUT_PREINITIALIZED`.
- If the ``pNext`` chain includes a ``VkExternalMemoryImageCreateInfo`` or ``VkExternalMemoryImageCreateInfoNV`` structure, its `externalMemoryProperties.compatibleHandleTypes` must contain the handle types specified in `handleTypes`.
- If the image format is one of those listed in Formats requiring sampler Yâ€™CBCR conversion for

- `VK_IMAGE_ASPECT_COLOR_BIT` image views, then `mipLevels` must be 1
- If the image format is one of those listed in Formats requiring sampler Yâ€™CBCR conversion for `VK_IMAGE_ASPECT_COLOR_BIT` image views, `samples` must be `VK_SAMPLE_COUNT_1_BIT`
- If the image format is one of those listed in Formats requiring sampler Yâ€™CBCR conversion for `VK_IMAGE_ASPECT_COLOR_BIT` image views, `imageType` must be `VK_IMAGE_TYPE_2D`
- If the image format is one of those listed in Formats requiring sampler Yâ€™CBCR conversion for `VK_IMAGE_ASPECT_COLOR_BIT` image views, and the `ycbcrImageArrays` feature is not enabled, `arrayLayers` must be 1
- 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`
- 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`
- If `tiling` is `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`, then the `pNext` chain must include exactly one of `VkImageDrmFormatModifierListCreateInfoEXT` or `VkImageDrmFormatModifierExplicitCreateInfoEXT` structures
- If the `pNext` chain includes a `VkImageDrmFormatModifierListCreateInfoEXT` or `VkImageDrmFormatModifierExplicitCreateInfoEXT` structure, then `tiling` must be `VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT`
- 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`.
- If `flags` contains `VK_IMAGE_CREATE_SAMPLE_LOCATIONS_COMPATIBLE_DEPTH_BIT_EXT` format must be a depth or depth/stencil format
- 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`.
- 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`.
- If the `pNext` chain includes a `VkExternalFormatANDROID` structure whose `externalFormat` member is not 0, `flags` must not include `VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT`.
- 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`.
- If the `pNext` chain includes a `VkExternalFormatANDROID` structure whose `externalFormat` member is not 0, `tiling` must be `VK_IMAGE_TILING_OPTIMAL`.
- 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`
- 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`
- 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`
- 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

- 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
- 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
- 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
- If flags contains VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV, imageType must be VK_IMAGE_TYPE_2D or VK_IMAGE_TYPE_3D
- 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
- 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
- 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
- If usage includes VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV, imageType must be VK_IMAGE_TYPE_2D.
- If usage includes VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV, samples must be VK_SAMPLE_COUNT_1_BIT.
- If usage includes VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV, tiling must be VK_IMAGE_TILING_OPTIMAL.
- If flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT, tiling must be VK_IMAGE_TILING_OPTIMAL
- If flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT, imageType must be VK_IMAGE_TYPE_2D
- If flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT, flags must not contain VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT
- If flags contains VK_IMAGE_CREATE_SUBSAMPLED_BIT_EXT, mipLevels must be 1

ImageView

Validations for VkImageViewCreateInfo:

- 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_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
- 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`
- The format features of the resultant image view must contain at least one bit.
- 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`.
- If `usage` contains `VK_IMAGE_USAGE_STORAGE_BIT`, then the image view's format features must contain `VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT`.
- If `usage` contains `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`, then the image view's format features must contain `VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT`.
- 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`.
- 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`.
- `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.baseMipLevel + subresourceRange.levelCount` must be less than or equal to the `mipLevels` specified in `VkImageCreateInfo` when image was created
- If image was created with `usage` containing `VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT`, `subresourceRange.levelCount` must be 1
- 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
- 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
- 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.
- 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.
- 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 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
- 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.
- 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.

- 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`.
- 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`
- 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
- 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`.
- 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`
- If image has an external format, format must be `VK_FORMAT_UNDEFINED`.
- 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.
- If image has an external format, all members of `components` must be `VK_COMPONENT_SWIZZLE_IDENTITY`.
- 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`
- If image was created with usage containing `VK_IMAGE_USAGE_SHADING_RATE_IMAGE_BIT_NV`, format must be `VK_FORMAT_R8_UINT`
- If dynamic fragment density map feature is not enabled, `flags` must not contain `VK_IMAGE_VIEW_CREATE_FRAGMENT_DENSITY_MAP_DYNAMIC_BIT_EXT`
- 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`
- 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
- 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
- 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`:

- If `levelCount` is not `VK_REMAINING_MIP_LEVELS`, it must be greater than 0
- If `layerCount` is not `VK_REMAINING_ARRAY_LAYERS`, it must be greater than 0
- 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`
- `aspectMask` must not include `VK_IMAGE_ASPECT_MEMORY_PLANE_i_BIT_EXT` for any index `i`