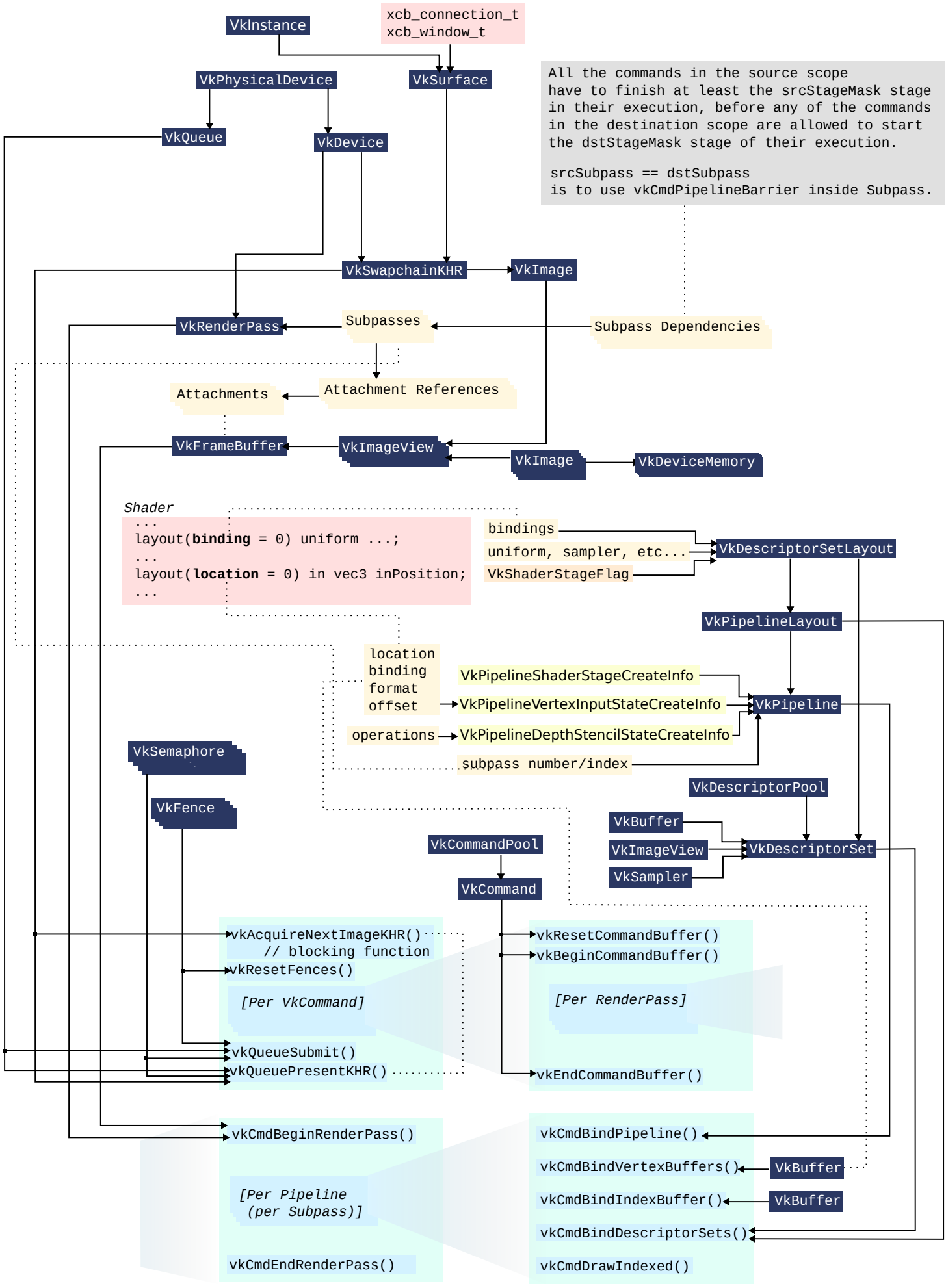
OVERVIEW


VkInstance
xcb_connection_t
xcb_window_t
VkPhysicalDevice
VkSurface
All the commands in the source scope
have to finish at least the srcStageMask stage
in their execution, before any of the commands
in the destination scope are allowed to start
the dstStageMask stage of their execution.
srcSubpass == dstSubpass
is to use vkCmdPipelineBarrier inside Subpass.
VkQueue
VkDevice
VkSwapchainKHR
VkImage
VkRenderPass
Subpasses
Subpass Dependencies
Attachments
Attachment References
VkFrameBuffer
VkImageView
VkImage
VkDeviceMemory
Shader
...
layout(binding = 0) uniform ...;
...
layout(location = 0) in vec3 inPosition;
...
bindings
uniform, sampler, etc...
VkShaderStageFlag
VkDescriptorSetLayout
VkPipelineLayout
location
binding
format
offset
VkPipelineShaderStageCreateInfo
VkPipelineVertexInputStateCreateInfo
VkPipeline
operations
VkPipelineDepthStencilStateCreateInfo
subpass number/index
VkSemaphore
VkFence
VkDescriptorPool
VkBuffer
VkImageView
VkSampler
VkDescriptorSet
VkCommandPool
VkCommand
vkAcquireNextImageKHR()
// blocking function
vkResetFences()
[Per VkCommand]
vkQueueSubmit()
vkQueuePresentKHR()
vkResetCommandBuffer()
vkBeginCommandBuffer()
[Per RenderPass]
vkEndCommandBuffer()
vkCmdBeginRenderPass()
[Per Pipeline
(per Subpass)]
vkCmdEndRenderPass()
vkCmdBindPipeline()
vkCmdBindVertexBuffers()
VkBuffer
vkCmdBindIndexBuffer()
VkBuffer
vkCmdBindDescriptorSets()
vkCmdDrawIndexed()

GLFW General

GLFWwindow* window;

glfwInit();

glfwWindowHint(GLFW_CLIENT_API, GLFW_NO_API);

window = **glfwCreateWindow**(<WIDTH>, <HEIGHT>, <WINDOW_NAME>, nullptr, nullptr);

glfwSetWindowUserPointer(window, <USER_DATA>);

glfwSetFramebufferSizeCallback(window, <USER_CALLBACK>);

static (void USER_CALLBACK*)(GLFWwindow* window, int width, int height);
// In this callback, just set an atomic flag to recrease the swap chain
// and the other resources.

bool **glfwWindowShouldClose**(window)

glfwGetFramebufferSize(window, &width, &height);

glfwWaitEventsTimeout(<SECONDS>);

glfwDestroyWindow(window);

glfwTerminate();

GLFW Vulkan Specific

int **glfwVulkanSupported**(); // GLFW_TRUE or GLFW_TRUE

int **glfwGetPhysicalDevicePresentationSupport** (instance, VkPhysicalDevice device, uint32_t queuefamily)
// queueFamily : index within the range returned by vkGetPhysicalDeviceQueueFamilyProperties().
// GLFW_TRUE / GLFW_FALSE

const char ** **glfwGetRequiredInstanceExtensions**(uint32_t * count)
// Ex.
// [0] "VK_KHR_surface"
// [1] "VK_KHR_xcb_surface"

VkResult **glfwCreateWindowSurface**(VkInstance instance, window, nullptr, **VkSurfaceKHR * surface**)

GLFWvkproc **glfwGetInstanceProcAddress** (instance, const char * procname)
// typedef void(* GLFWvkproc) (void)
// instance can be nullptr
// procname is something like "vkDestroyImageView"

VkSwapchainCreateInfoKHR createInfo{}
createInfo.surface = surface;
...
vkCreateSwapchainKHR(..., &createInfo, ...);

vkGetPhysicalDeviceSurfaceSupportKHR(..., VkBool32* pSupported)

vkGetPhysicalDeviceSurfaceCapabilitiesKHR( )

```
typedef struct VkSurfaceCapabilitiesKHR {
    uint32_t                         minImageCount; // 2
    uint32_t                         maxImageCount; // 8
    VkExtent2D                       currentExtent;  // (800, 600)
    VkExtent2D                       minImageExtent; // (800, 600)
    VkExtent2D                       maxImageExtent; // (800, 600)
    uint32_t                         maxImageArrayLayers; // 1
    VkSurfaceTransformFlagsKHR       supportedTransforms; // VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR
    VkSurfaceTransformFlagBitsKHR    currentTransform; // VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR
    VkCompositeAlphaFlagsKHR         supportedCompositeAlpha; // VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR
    VkImageUsageFlags                supportedUsageFlags; // VK_IMAGE_USAGE_TRANSFER_SRC_BIT
} VkSurfaceCapabilitiesKHR;                               // VK_IMAGE_USAGE_TRANSFER_DST_BIT
                                                          // VK_IMAGE_USAGE_SAMPLED_BIT
                                                          // VK_IMAGE_USAGE_STORAGE_BIT
                                                          // VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
                                                          // VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
```

vkGetPhysicalDeviceSurfacePresentModesKHR()
// VK_PRESENT_MODE_IMMEDIATE_KHR = 0,
// VK_PRESENT_MODE_FIFO_KHR = 2,
// VK_PRESENT_MODE_FIFO_RELAXED_KHR = 3,

vkGetPhysicalDeviceSurfaceFormatsKHR( )

```
typedef struct VkSurfaceFormatKHR {
    VkFormat           format; // VK_FORMAT_B8G8R8A8_UNORM
                               // VK_FORMAT_B8G8R8A8_SRGB
    VkColorSpaceKHR    colorSpace; // VK_COLOR_SPACE_SRGB_NONLINEAR_KHR
} VkSurfaceFormatKHR;
```

VkInstance

```
// Returns global extension properties.
VkResult vkEnumerateInstanceExtensionProperties(
    const char* pLayerName, // usually nullptr
    uint32_t* pPropertyCount,
    VkExtensionProperties* pProperties
)
```

VkExtensionProperties

```
char extensionName[VK_MAX_EXTENSION_NAME_SIZE]
    // 256 in vulkan_core.h
uint32_t   specVersion
```

Ex.

```
VK_KHR_device_group_creation    1
VK_KHR_display  23
VK_KHR_external_fence_capabilities      1
VK_KHR_external_memory_capabilities     1
VK_KHR_external_semaphore_capabilities  1
VK_KHR_get_display_properties2  1
VK_KHR_get_physical_device_properties2  2
VK_KHR_get_surface_capabilities2        1
VK_KHR_surface  25
VK_KHR_surface_protected_capabilities   1
VK_KHR_wayland_surface  6
VK_KHR_xcb_surface      6
VK_KHR_xlib_surface     6
VK_EXT_acquire_drm_display      1
VK_EXT_acquire_xlib_display     1
VK_EXT_debug_report     10
VK_EXT_direct_mode_display      1
VK_EXT_display_surface_counter  1
VK_EXT_debug_utils      2
VK_KHR_portability_enumeration  1
```

```
const char ** glfwGetRequiredInstanceExtensions( uint32_t * count )
```

Ex.

```
[0] "VK_KHR_surface"
[1] "VK_KHR_xcb_surface"
```

VkApplicationInfo

```
sType = VK_STRUCTURE_TYPE_APPLICATION_INFO
pApplicationName = <APP_NAME>
applicationVersion = VK_MAKE_VERSION(1, 0, 0)
pEngineName = <USER_ENGINE_NAME>
engineVersion = VK_MAKE_VERSION(1, 0, 0)
apiVersion = VK_API_VERSION_1_0 // Minimum Version that App uses
```

```
VKAPI_ATTR (VkBool32 VKAPI_CALL *)(
    VkDebugUtilsMessageSeverityFlagBitsEXT messageSeverity,
    VkDebugUtilsMessageTypeFlagsEXT messageType,
    const VkDebugUtilsMessengerCallbackDataEXT* pCallbackData,
    void* pUserData
) // returning true aborts the program.
```

VkDebugUtilsMessengerCreateInfoEXT

```
sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT
messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_VERBOSE_BIT_EXT
                  VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT
                | VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT
messageType = VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT
            | VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT
            | VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT
pfnUserCallback
```

VkInstanceCreateInfo

```
sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO
pApplicationInfo
enabledExtensionCount
ppEnabledExtensionNames
enabledLayerCount = 1
    // or 0 if the validation layer is not needed.
ppEnabledLayerNames
    = (const char* const*)"VK_LAYER_KHRONOS_validation"
pNext
```

```
VkResult vkCreateInstance(
    const VkInstanceCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, // usually nullptr
    VkInstance* pInstance
)
```

→ VkInstance

```
void vkDestroyInstance(
    VkInstance instance,
    const VkAllocationCallbacks* pAllocator // usually nullptr
)
```

VkInstance →

VkSurface

```
VkResult glfwCreateWindowSurface  (
    VkInstance                   instance,
    GLFWwindow*                  window,
    const VkAllocationCallbacks* allocator,
    VkSurfaceKHR*                surface
)
```

VkInstance

VkSurfaceKHR

```
VkXcbSurfaceCreateInfoKHR{
sType = VK_STRUCTURE_TYPE_XCB_SURFACE_CREATE_INFO_KHR;
pNext = nullptr;
flags = 0;
xcb_connection_t* connection;
xcb_window_t      window;
```

```
VkResult vkCreateXcbSurfaceKHR(
    VkInstance                         instance,
    const VkXcbSurfaceCreateInfoKHR*   pCreateInfo,
    const VkAllocationCallbacks*       pAllocator,
    VkSurfaceKHR*                      pSurface
);
```

VkInstance

```
void vkDestroySurfaceKHR(
    VkInstance                   instance,
    VkSurfaceKHR                 surface,
    const VkAllocationCallbacks* pAllocator // usually nullptr
);
```

VkInstance

VkSurfaceKHR

VkPhysicalDevice, VkExtensionProperties, VkSurfaceCapabilitiesKHR, VkSurfaceFormatKHR, and VkPresentModeKHR

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

VkInstance

VkPhysicalDevice

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

VkPhysicalDevice

VkExtensionProperties

```
char extensionName[VK_MAX_EXTENSION_NAME_SIZE] // 256 in vulkan_core.h
uint32_t  specVersion
        VK_KHR_16bit_storage 1
        ...
        VK_KHR_swapchain 70
        VK_KHR_zero_initialize_workgroup_memory 1
        VK_EXT_4444_formats 1
        ...
        VK_EXT_ycbcr_image_arrays 1
        VK_NV_clip_space_w_scaling 1
        ...
        VK_NV_viewport_swizzle 1
        VK_NVX_binary_import 1
        VK_NVX_image_view_handle 2
        VK_NVX_multiview_per_view_attributes 1
```

```
VkResult vkGetPhysicalDeviceSurfaceCapabilitiesKHR(
    VkPhysicalDevice           physicalDevice,
    VkSurfaceKHR               surface,
    VkSurfaceCapabilitiesKHR* pSurfaceCapabilities
)
```

VkPhysicalDevice

VkSurfaceKHR

VkSurfaceCapabilitiesKHR

```
uint32_t minImageCount; // 2
uint32_t maxImageCount; // 8
VkExtent2D currentExtent; // (800, 600) depends on the current window
VkExtent2D minImageExtent; // (800, 600) depends on the current window
VkExtent2D maxImageExtent; // (800, 600) depends on the current window
uint32_t maxImageArrayLayers;  // 1
VkSurfaceTransformFlagsKHR supportedTransforms; // VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR
VkSurfaceTransformFlagBitsKHR currentTransform; // VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR
VkCompositeAlphaFlagsKHR supportedCompositeAlpha; // VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR
VkImageUsageFlags supportedUsageFlags; // VK_IMAGE_USAGE_TRANSFER_SRC_BIT
                                       // VK_IMAGE_USAGE_TRANSFER_DST_BIT
                                       // VK_IMAGE_USAGE_SAMPLED_BIT
                                       // VK_IMAGE_USAGE_STORAGE_BIT
                                       // VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
                                       // VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
```

```
VkResult vkGetPhysicalDeviceSurfaceFormatsKHR(
    VkPhysicalDevice    physicalDevice,
    VkSurfaceKHR        surface,
    uint32_t*           pSurfaceFormatCount,
    VkSurfaceFormatKHR* pSurfaceFormats);
```

VkPhysicalDevice

VkSurfaceKHR

VkSurfaceFormatKHR

```
VkFormat        format;
    // VK_FORMAT_B8G8R8A8_UNORM = 44
    // VK_FORMAT_B8G8R8A8_SRGB = 50,
VkColorSpaceKHR colorSpace;
    // VK_COLOR_SPACE_SRGB_NONLINEAR_KHR = 0,
```

```
VkResult vkGetPhysicalDeviceSurfacePresentModesKHR(
    VkPhysicalDevice  physicalDevice,
    VkSurfaceKHR      surface,
    uint32_t*         pPresentModeCount,
    VkPresentModeKHR* pPresentModes
        // VK_PRESENT_MODE_IMMEDIATE_KHR = 0,
        // (VK_PRESENT_MODE_MAILBOX_KHR = 1,)
        // VK_PRESENT_MODE_FIFO_KHR = 2,
        // VK_PRESENT_MODE_FIFO_RELAXED_KHR = 3,
);
```

VkPhysicalDevice

VkSurfaceKHR

typedef enum VkPresentModeKHR {

```
VK_PRESENT_MODE_IMMEDIATE_KHR = 0,
VK_PRESENT_MODE_MAILBOX_KHR = 1,
VK_PRESENT_MODE_FIFO_KHR = 2,
VK_PRESENT_MODE_FIFO_RELAXED_KHR = 3,
VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR,
VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR,
} VkPresentModeKHR;
```

VkPhysicalDeviceProperties & VkPhysicalDeviceFeatures

```
void vkGetPhysicalDeviceProperties(
    VkPhysicalDevice           physicalDevice,
    VkPhysicalDeviceProperties* pProperties
);
```

VkPhysicalDevice

VkPhysicalDeviceProperties

```
uint32_t                         apiVersion; //4206797
uint32_t                         driverVersion; //2182037824
uint32_t                         vendorID; // 4318
uint32_t                         deviceID; // 9479
VkPhysicalDeviceType             deviceType; // VK_PHYSICAL_DEVICE_TYPE_DISCRETE_GPU = 2,
char                             deviceName[VK_MAX_PHYSICAL_DEVICE_NAME_SIZE=16];
                                     // NVIDIA GeForce RTX 3050
uint8_t                          pipelineCacheUUID[VK_UUID_SIZE];
                                     // 7bc3bbe9d284d0cb29162032ef3d4e8e
VkPhysicalDeviceLimits           limits;
VkPhysicalDeviceSparseProperties sparseProperties;
```

VkPhysicalDeviceLimits

```
maxImageDimension1D: 32768
maxImageDimension2D: 32768
maxImageDimension3D: 16384
maxImageDimensionCube: 32768
maxImageArrayLayers: 2048
maxTexelBufferElements: 134217728
maxUniformBufferRange: 65536
maxStorageBufferRange: 4294967295
maxPushConstantsSize: 256
maxMemoryAllocationCount: 4294967295
maxSamplerAllocationCount: 4000
bufferImageGranularity: 1024
sparseAddressSpaceSize: 1099511627775
maxBoundDescriptorSets: 32
maxPerStageDescriptorSamplers: 1048576
maxPerStageDescriptorUniformBuffers: 1048576
maxPerStageDescriptorStorageBuffers: 1048576
maxPerStageDescriptorSampledImages: 1048576
maxPerStageDescriptorStorageImages: 1048576
maxPerStageDescriptorInputAttachments: 1048576
maxPerStageResources: 4294967295
maxDescriptorSetSamplers: 1048576
maxDescriptorSetUniformBuffers: 1048576
maxDescriptorSetUniformBuffersDynamic: 15
maxDescriptorSetStorageBuffers: 1048576
maxDescriptorSetStorageBuffersDynamic: 16
maxDescriptorSetSampledImages: 1048576
maxDescriptorSetStorageImages: 1048576
maxDescriptorSetInputAttachments: 1048576
maxVertexInputAttributes: 32
maxVertexInputBindings: 32
maxVertexInputAttributeOffset: 2047
maxVertexInputBindingStride: 2048
maxVertexOutputComponents: 128
maxTessellationGenerationLevel: 64
maxTessellationPatchSize: 32
maxTessellationControlPerVertexInputComponents: 128
maxTessellationControlPerVertexOutputComponents: 128
maxTessellationControlPerPatchOutputComponents: 120
maxTessellationControlTotalOutputComponents: 4216
maxTessellationEvaluationInputComponents: 128
maxTessellationEvaluationOutputComponents: 128
maxGeometryShaderInvocations: 32
maxGeometryInputComponents: 128
maxGeometryOutputComponents: 128
maxGeometryOutputVertices: 1024
maxGeometryTotalOutputComponents: 1024
maxFragmentInputComponents: 128
maxFragmentOutputAttachments: 8
maxFragmentDualSrcAttachments: 1
maxFragmentCombinedOutputResources: 4294967295
maxComputeSharedMemorySize: 49152
maxComputeWorkGroupCount: (2147483647, 65535, 65535)
maxComputeWorkGroupInvocations: 1024
maxComputeWorkGroupSize: (1024, 1024, 64)
subPixelPrecisionBits: 8
subTexelPrecisionBits: 8
mipmapPrecisionBits: 8
maxDrawIndexedIndexValue: 4294967295
maxDrawIndirectCount: 4294967295
maxSamplerLodBias: 15
maxSamplerAnisotropy: 16
maxViewports: 16
maxViewportDimensions: (32768, 32768)
viewportBoundsRange: (-65536, 65536)
viewportSubPixelBits: 8
minMemoryMapAlignment: 64
minTexelBufferOffsetAlignment: 16
minUniformBufferOffsetAlignment: 64
minStorageBufferOffsetAlignment: 16
minTexelOffset: -8
maxTexelOffset: 7
minTexelGatherOffset: -32
maxTexelGatherOffset: 31
minInterpolationOffset: -0.5
maxInterpolationOffset: 0.4375
subPixelInterpolationOffsetBits: 4
maxFramebufferWidth: 32768
maxFramebufferHeight: 32768
maxFramebufferLayers: 2048
framebufferColorSampleCounts: 0xf
framebufferDepthSampleCounts: 0xf
framebufferStencilSampleCounts: 0x1f
framebufferNoAttachmentsSampleCounts: 0x1f
maxColorAttachments: 8
sampledImageColorSampleCounts: 0xf
sampledImageIntegerSampleCounts: 0xf
sampledImageDepthSampleCounts: 0xf
sampledImageStencilSampleCount: 0x1f
storageImageSampleCounts: 0xf
maxSampleMaskWords: 1
timestampComputeAndGraphics: 1
timestampPeriod: 1
maxClipDistances: 8
maxCullDistances: 8
maxCombinedClipAndCullDistances: 8
discreteQueuePriorities: 2
pointSizeRange: (1, 2047.94)
lineWidthRange: (1, 64)
pointSizeGranularity: 0.0625
lineWidthGranularity: 0.0625
strictLines: 1
standardSampleLocations: 1
optimalBufferCopyOffsetAlignment: 1
optimalBufferCopyRowPitchAlignment: 1
nonCoherentAtomSize: 64
```

VkPhysicalDeviceSparseProperties

```
    VkBool32    residencyStandard2DBlockShape; // 1
    VkBool32    residencyStandard2DMultisampleBlockShape; // 1
    VkBool32    residencyStandard3DBlockShape; // 1
    VkBool32    residencyAlignedMipSize; // 0
    VkBool32    residencyNonResidentStrict; // 1
```

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

VkPhysicalDevice

VkPhysicalDeviceFeatures

```
robustBufferAccess: 1
fullDrawIndexUint32: 1
imageCubeArray: 1
independentBlend: 1
geometryShader: 1
tessellationShader: 1
sampleRateShading: 1
dualSrcBlend: 1
logicOp: 1
multiDrawIndirect: 1
drawIndirectFirstInstance: 1
depthClamp: 1
depthBiasClamp: 1
fillModeNonSolid: 1
depthBounds: 1
wideLines: 1
largePoints: 1
alphaToOne: 1
multiViewport: 1
samplerAnisotropy: 1
textureCompressionETC2: 0
textureCompressionASTC_LDR: 0
textureCompressionBC: 1
occlusionQueryPrecise: 1
pipelineStatisticsQuery: 1
vertexPipelineStoresAndAtomics: 1
fragmentStoresAndAtomics: 1
shaderTessellationAndGeometryPointSize: 1
shaderImageGatherExtended: 1
shaderStorageImageExtendedFormats: 1
shaderStorageImageMultisample: 1
shaderStorageImageReadWithoutFormat: 1
shaderStorageImageWriteWithoutFormat: 1
shaderUniformBufferArrayDynamicIndexing: 1
shaderSampledImageArrayDynamicIndexing: 1
shaderStorageBufferArrayDynamicIndexing: 1
shaderStorageImageArrayDynamicIndexing: 1
shaderClipDistance: 1
shaderCullDistance: 1
shaderFloat64: 1
shaderInt64: 1
shaderInt16: 1
shaderResourceResidency: 1
shaderResourceMinLod: 1
sparseBinding: 1
sparseResidencyBuffer: 1
sparseResidencyImage2D: 1
sparseResidencyImage3D: 1
sparseResidency2Samples: 1
sparseResidency4Samples: 1
sparseResidency8Samples: 1
sparseResidency16Samples: 1
sparseResidencyAliased: 1
variableMultisampleRate: 1
inheritedQueries: 1
```

VkPhysicalDeviceMemoryProperties

```
void vkGetPhysicalDeviceMemoryProperties(
    VkPhysicalDevice                    physicalDevice,
    VkPhysicalDeviceMemoryProperties*   pMemoryProperties
);
```

VkPhysicalDevice

```
typedef struct VkPhysicalDeviceMemoryProperties {
    uint32_t        memoryTypeCount;
    VkMemoryType    memoryTypes[VK_MAX_MEMORY_TYPES];
    uint32_t        memoryHeapCount;
    VkMemoryHeap    memoryHeaps[VK_MAX_MEMORY_HEAPS];
} VkPhysicalDeviceMemoryProperties;
```

```
typedef struct VkMemoryType {
    VkMemoryPropertyFlags    propertyFlags;
    uint32_t                 heapIndex;
} VkMemoryType;
```

```
typedef struct VkMemoryHeap {
    VkDeviceSize         size;
    VkMemoryHeapFlags    flags;
} VkMemoryHeap;
```

```
typedef enum VkMemoryPropertyFlagBits {
    VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT = 0x00000001,
    VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT = 0x00000002,
    VK_MEMORY_PROPERTY_HOST_COHERENT_BIT = 0x00000004,
    VK_MEMORY_PROPERTY_HOST_CACHED_BIT = 0x00000008,
    VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT = 0x00000010,
  // Provided by VK_VERSION_1_1
    VK_MEMORY_PROPERTY_PROTECTED_BIT = 0x00000020,
  // Provided by VK_AMD_device_coherent_memory
    VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD = 0x00000040,
  // Provided by VK_AMD_device_coherent_memory
    VK_MEMORY_PROPERTY_DEVICE_UNCACHED_BIT_AMD = 0x00000080,
  // Provided by VK_NV_external_memory_rdma
    VK_MEMORY_PROPERTY_RDMA_CAPABLE_BIT_NV = 0x00000100,
} VkMemoryPropertyFlagBits;
```

```
typedef enum VkMemoryHeapFlagBits {
    VK_MEMORY_HEAP_DEVICE_LOCAL_BIT = 0x00000001,
  // Provided by VK_VERSION_1_1
    VK_MEMORY_HEAP_MULTI_INSTANCE_BIT = 0x00000002,
  // Provided by VK_KHR_device_group_creation
    VK_MEMORY_HEAP_MULTI_INSTANCE_BIT_KHR
        = VK_MEMORY_HEAP_MULTI_INSTANCE_BIT,
} VkMemoryHeapFlagBits;
```

Example from NVIDIA Geforce RTX 3060 8GB KFA2

```
memoryTypeCount: 5

memoryTypes[0].heapIndex: 1
memoryTypes[0].propertyFlags:
memoryTypes[1].heapIndex: 0
memoryTypes[1].propertyFlags: VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT
memoryTypes[2].heapIndex: 1
memoryTypes[2].propertyFlags: VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT
                              VK_MEMORY_PROPERTY_HOST_COHERENT_BIT
memoryTypes[3].heapIndex: 1
memoryTypes[3].propertyFlags: VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT
                              VK_MEMORY_PROPERTY_HOST_COHERENT_BIT
                              VK_MEMORY_PROPERTY_HOST_CACHED_BIT
memoryTypes[4].heapIndex: 2
memoryTypes[4].propertyFlags: VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT
                              VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT
                              VK_MEMORY_PROPERTY_HOST_COHERENT_BIT
memoryHeapCount: 3

memoryHeaps[0].size 8192[MB]
memoryHeaps[0].flags: VK_MEMORY_HEAP_DEVICE_LOCAL_BIT
memoryHeaps[1].size 24016[MB]
memoryHeaps[1].flags:
memoryHeaps[2].size 246[MB]
memoryHeaps[2].flags: VK_MEMORY_HEAP_DEVICE_LOCAL_BIT
```

VkDevice & VkQueue

```
void vkGetPhysicalDeviceQueueFamilyProperties(
    VkPhysicalDevice         physicalDevice,
    uint32_t*                pQueueFamilyPropertyCount,
    VkQueueFamilyProperties* pQueueFamilyProperties
);
```

VkPhysicalDevice

VkQueueFamilyProperties

```
VkQueueFlags queueFlags;
    // VK_QUEUE_GRAPHICS_BIT
    // VK_QUEUE_COMPUTE_BIT
    // VK_QUEUE_TRANSFER_BIT
    // VK_QUEUE_SPARSE_BINDING_BIT
    // VK_QUEUE_PROTECTED_BIT
    // VK_QUEUE_OPTICAL_FLOW_BIT_NV
uint32_t     queueCount; // 16
uint32_t     timestampValidBits; // 64
VkExtent3D   minImageTransferGranularity; // (1, 1, 1)
```

```
VkResult vkGetPhysicalDeviceSurfaceSupportKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t         queueFamilyIndex,
    VkSurfaceKHR     surface,
    VkBool32*        pSupported
);
```

VkPhysicalDevice

VkSurfaceKHR

VkDeviceCreateInfo

```
sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
pNext; // usually nullptr
flags; // usually 0.
queueCreateInfoCount; // number of queue family indices.
pQueueCreateInfos; // (const VkDeviceQueueCreateInfo*)
enabledLayerCount = 1; // or 0 if the validation layer is not needed.
ppEnabledLayerNames;(const char* const*)"VK_LAYER_KHRONOS_validation"
enabledExtensionCount; // 1
ppEnabledExtensionNames
    // VK_KHR_SWAPCHAIN_EXTENSION_NAME ("VK_KHR_swapchain")
pEnabledFeatures;
```

VkPhysicalDeviceFeatures

```
...
samplerAnisotropy = VK_TRUE;
...
```

VkDeviceQueueCreateInfo

```
sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queueFamilyIndex = <QUEUE_FAMILY_INDEX>;
queueCount = <NUM QUEUES>;
pQueuePriorities = &queuePriority;// 0.0-1.0
pNext; // usually nullptr
flags; // usually = 0
```

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

VkPhysicalDevice

VkDevice

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

VkDevice

```
void vkGetDeviceQueue(
    VkDevice device,
    uint32_t queueFamilyIndex,
    uint32_t queueIndex,// must be within the range specified to queueCount
                        // in VkDeviceQueueCreateInfo given to vkCreateDevice
                        // for the queue family index.
    VkQueue* pQueue
);
```

VkPhysicalDevice

VkQueue

vkGetPhysicalDeviceSurfaceSupportKHR()
vkGetPhysicalDeviceSurfaceCapabilitiesKHR()
vkGetPhysicalDeviceSurfacePresentModesKHR()
vkGetPhysicalDeviceSurfaceFormatsKHR()
for VkSwapchainKHR

```
VkResult vkGetPhysicalDeviceSurfaceSupportKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t         queueFamilyIndex,
    VkSurfaceKHR     surface,
    VkBool32*        pSupported
);
```

VkPhysicalDevice

VkSurfaceKHR

```
VkResult vkGetPhysicalDeviceSurfaceCapabilitiesKHR(
    VkPhysicalDevice          physicalDevice,
    VkSurfaceKHR              surface,
    VkSurfaceCapabilitiesKHR* pSurfaceCapabilities
);
```

VkPhysicalDevice

VkSurfaceKHR

```
typedef struct VkSurfaceCapabilitiesKHR {
minImageCount; // Ex. 2
maxImageCount; // Ex. 8
currentExtent;  // Ex. (800, 600)
minImageExtent; // Ex. (800, 600)
maxImageExtent; // Ex. (800, 600)
maxImageArrayLayers; // Ex. 1
supportedTransforms;
currentTransform;
supportedCompositeAlpha;
supportedUsageFlags;
} VkSurfaceCapabilitiesKHR;
```

```
typedef enum VkSurfaceTransformFlagBitsKHR {
    VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR = 0x00000001,
    VK_SURFACE_TRANSFORM_ROTATE_90_BIT_KHR = 0x00000002,
    VK_SURFACE_TRANSFORM_ROTATE_180_BIT_KHR = 0x00000004,
    VK_SURFACE_TRANSFORM_ROTATE_270_BIT_KHR = 0x00000008,
    VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_BIT_KHR = 0x00000010,
    VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_90_BIT_KHR,
    VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_180_BIT_KHR,
    VK_SURFACE_TRANSFORM_HORIZONTAL_MIRROR_ROTATE_270_BIT_KHR,
    VK_SURFACE_TRANSFORM_INHERIT_BIT_KHR = 0x00000100,
} VkSurfaceTransformFlagBitsKHR;
```

```
typedef enum VkCompositeAlphaFlagBitsKHR {
    VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR = 0x00000001,
    VK_COMPOSITE_ALPHA_PRE_MULTIPLIED_BIT_KHR = 0x00000002,
    VK_COMPOSITE_ALPHA_POST_MULTIPLIED_BIT_KHR = 0x00000004,
    VK_COMPOSITE_ALPHA_INHERIT_BIT_KHR = 0x00000008,
} VkCompositeAlphaFlagBitsKHR;
```

```
typedef enum VkImageUsageFlagBits {
    VK_IMAGE_USAGE_TRANSFER_SRC_BIT = 0x00000001,
    VK_IMAGE_USAGE_TRANSFER_DST_BIT = 0x00000002,
    VK_IMAGE_USAGE_SAMPLED_BIT = 0x00000004,
    VK_IMAGE_USAGE_STORAGE_BIT = 0x00000008,
    VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT = 0x00000010,
    VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT = 0x00000020,
    VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT = 0x00000040,
    VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT = 0x00000080,
    ...
} VkImageUsageFlagBits;
```

```
typedef struct VkExtent2D {
    uint32_t    width;
    uint32_t    height;
} VkExtent2D;
```

```
VkResult vkGetPhysicalDeviceSurfacePresentModesKHR(
    VkPhysicalDevice  physicalDevice,
    VkSurfaceKHR      surface,
    uint32_t*         pPresentModeCount,
    VkPresentModeKHR* PresentModes
);
```

VkPhysicalDevice

VkSurfaceKHR

```
typedef enum VkPresentModeKHR {
    VK_PRESENT_MODE_IMMEDIATE_KHR = 0,
    VK_PRESENT_MODE_MAILBOX_KHR = 1,
    VK_PRESENT_MODE_FIFO_KHR = 2,
    VK_PRESENT_MODE_FIFO_RELAXED_KHR = 3,
  // Provided by VK_KHR_shared_presentable_image
    VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR = 1000111000,
  // Provided by VK_KHR_shared_presentable_image
    VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR = 1000111001,
} VkPresentModeKHR;
```

```
VkResult vkGetPhysicalDeviceSurfaceFormatsKHR(
    VkPhysicalDevice    physicalDevice,
    VkSurfaceKHR        surface,
    uint32_t*           pSurfaceFormatCount,
    VkSurfaceFormatKHR* pSurfaceFormats
);
```

VkPhysicalDevice

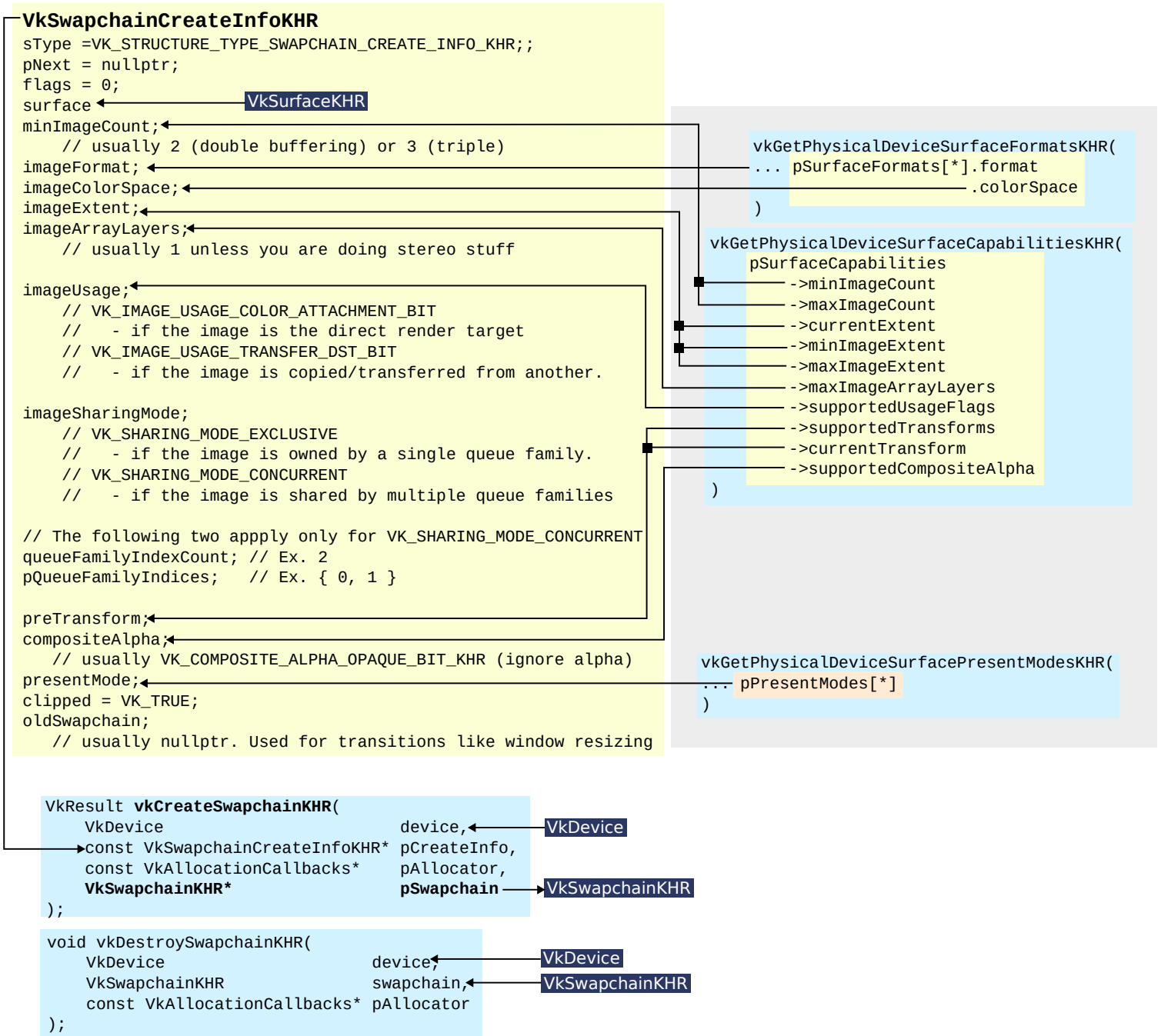
VkSurfaceKHR

```
typedef struct VkSurfaceFormatKHR {
    VkFormat           format;
    VkColorSpaceKHR    colorSpace;
} VkSurfaceFormatKHR;
```

```
typedef enum VkFormat {
    ...
    VK_FORMAT_R8G8B8_UNORM = 23,
    ...
    VK_FORMAT_R8G8B8_SRGB = 29,
    ...
} VkFormat;
```

```
typedef enum VkColorSpaceKHR {
    VK_COLOR_SPACE_SRGB_NONLINEAR_KHR,
    VK_COLOR_SPACE_DISPLAY_P3_NONLINEAR_EXT,
    VK_COLOR_SPACE_EXTENDED_SRGB_LINEAR_EXT,
    VK_COLOR_SPACE_DISPLAY_P3_LINEAR_EXT,
    VK_COLOR_SPACE_DCI_P3_NONLINEAR_EXT,
    VK_COLOR_SPACE_BT709_LINEAR_EXT,
    VK_COLOR_SPACE_BT709_NONLINEAR_EXT,
    VK_COLOR_SPACE_BT2020_LINEAR_EXT,
    VK_COLOR_SPACE_HDR10_ST2084_EXT,
    VK_COLOR_SPACE_DOLBYVISION_EXT,
    VK_COLOR_SPACE_HDR10_HLG_EXT,
    VK_COLOR_SPACE_ADOBERGB_LINEAR_EXT,
    VK_COLOR_SPACE_ADOBERGB_NONLINEAR_EXT,
    VK_COLOR_SPACE_PASS_THROUGH_EXT,
    VK_COLOR_SPACE_EXTENDED_SRGB_NONLINEAR_EXT,
    VK_COLOR_SPACE_DISPLAY_NATIVE_AMD,
    VK_COLORSPACE_SRGB_NONLINEAR_KHR,
    VK_COLOR_SPACE_DCI_P3_LINEAR_EXT,
} VkColorSpaceKHR;
```

VkSwapchainKHR




```
VkResult vkCreateSwapchainKHR(
    VkDevice                        device,
    const VkSwapchainCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks*    pAllocator,
    VkSwapchainKHR*                 pSwapchain
);
```

VkDevice

VkSwapchainKHR

```
void vkDestroySwapchainKHR(
    VkDevice                    device,
    VkSwapchainKHR              swapchain,
    const VkAllocationCallbacks* pAllocator
);
```

VkDevice

VkSwapchainKHR

VkSubpassDescription for VkRenderPass

```
typedef enum VkPipelineBindPoint {
    VK_PIPELINE_BIND_POINT_GRAPHICS = 0,
    VK_PIPELINE_BIND_POINT_COMPUTE = 1,
  // Provided by VK_KHR_ray_tracing_pipeline
    VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR = 1000165000,
  // Provided by VK_HUAWEI_subpass_shading
    VK_PIPELINE_BIND_POINT_SUBPASS_SHADING_HUAWEI = 1000369003,
  // Provided by VK_NV_ray_tracing
    VK_PIPELINE_BIND_POINT_RAY_TRACING_NV = VK_PIPELINE_BIND_POINT_RAY_TRACING_KHR,
} VkPipelineBindPoint;
```

```
typedef enum VkImageLayout {
    VK_IMAGE_LAYOUT_UNDEFINED = 0,
    VK_IMAGE_LAYOUT_GENERAL = 1,
    VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL = 2,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL = 3,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL = 4,
    VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL = 5,
    VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL = 6,
    VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL = 7,
    VK_IMAGE_LAYOUT_PREINITIALIZED = 8,
    ...
} VkImageLayout;
```

```
index in VkRenderPassCreateInfo::pAttachments,
or VK_ATTACHMENT_UNUSED
```

```
typedef struct VkAttachmentReference {
    uint32_t         attachment;
    VkImageLayout    layout;
} VkAttachmentReference;
```

```
typedef struct VkSubpassDescription {
    VkSubpassDescriptionFlags       flags; // usually 0
    VkPipelineBindPoint             pipelineBindPoint;
    uint32_t                        inputAttachmentCount;
    const VkAttachmentReference*    pInputAttachments;
    uint32_t                        colorAttachmentCount;
    const VkAttachmentReference*    pColorAttachments;
    const VkAttachmentReference*    pResolveAttachments;
    const VkAttachmentReference*    pDepthStencilAttachment;
    uint32_t                        preserveAttachmentCount;
    const uint32_t*                 pPreserveAttachments;
} VkSubpassDescription;
```

VkSubpassDependency for VkRenderPass

```
typedef enum VkPipelineStageFlagBits {
    VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT = 0x00000001,
    VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT = 0x00000002,
    VK_PIPELINE_STAGE_VERTEX_INPUT_BIT = 0x00000004,
    VK_PIPELINE_STAGE_VERTEX_SHADER_BIT = 0x00000008,
    VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT = 0x00000010,
    VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT = 0x00000020,
    VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT = 0x00000040,
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT = 0x00000080,
    VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT = 0x00000100,
    VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT = 0x00000200,
    VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT = 0x00000400,
    VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT = 0x00000800,
    VK_PIPELINE_STAGE_TRANSFER_BIT = 0x00001000,
    VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT = 0x00002000,
    VK_PIPELINE_STAGE_HOST_BIT = 0x00004000,
    VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT = 0x00008000,
    VK_PIPELINE_STAGE_ALL_COMMANDS_BIT = 0x00010000,
    // Provided by VK_VERSION_1_3
      VK_PIPELINE_STAGE_NONE = 0,
    // Provided by VK_EXT_transform_feedback
      VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT = 0x01000000,
    // Provided by VK_EXT_conditional_rendering
      VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT = 0x00040000,
    // Provided by VK_KHR_acceleration_structure
      VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR = 0x02000000,
    // Provided by VK_KHR_ray_tracing_pipeline
      VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR = 0x00200000,
    // Provided by VK_EXT_fragment_density_map
      VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT = 0x00800000,
    // Provided by VK_KHR_fragment_shading_rate
      VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR = 0x00400000,
    // Provided by VK_NV_device_generated_commands
      VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV = 0x00020000,
    // Provided by VK_EXT_mesh_shader
      VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT = 0x00080000,
    // Provided by VK_EXT_mesh_shader
      VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT = 0x00100000,
    // Provided by VK_NV_shading_rate_image
      VK_PIPELINE_STAGE_SHADING_RATE_IMAGE_BIT_NV = VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR,
    // Provided by VK_NV_ray_tracing
      VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_NV = VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR,
    // Provided by VK_NV_ray_tracing
      VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_NV = VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR,
    // Provided by VK_NV_mesh_shader
      VK_PIPELINE_STAGE_TASK_SHADER_BIT_NV = VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT,
    // Provided by VK_NV_mesh_shader
      VK_PIPELINE_STAGE_MESH_SHADER_BIT_NV = VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT,
    // Provided by VK_KHR_synchronization2
      VK_PIPELINE_STAGE_NONE_KHR = VK_PIPELINE_STAGE_NONE,
} VkPipelineStageFlagBits;
```

```
typedef enum VkAccessFlagBits {
    VK_ACCESS_INDIRECT_COMMAND_READ_BIT = 0x00000001,
    VK_ACCESS_INDEX_READ_BIT = 0x00000002,
    VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT = 0x00000004,
    VK_ACCESS_UNIFORM_READ_BIT = 0x00000008,
    VK_ACCESS_INPUT_ATTACHMENT_READ_BIT = 0x00000010,
    VK_ACCESS_SHADER_READ_BIT = 0x00000020,
    VK_ACCESS_SHADER_WRITE_BIT = 0x00000040,
    VK_ACCESS_COLOR_ATTACHMENT_READ_BIT = 0x00000080,
    VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT = 0x00000100,
    VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT = 0x00000200,
    VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT = 0x00000400,
    VK_ACCESS_TRANSFER_READ_BIT = 0x00000800,
    VK_ACCESS_TRANSFER_WRITE_BIT = 0x00001000,
    VK_ACCESS_HOST_READ_BIT = 0x00002000,
    VK_ACCESS_HOST_WRITE_BIT = 0x00004000,
    VK_ACCESS_MEMORY_READ_BIT = 0x00008000,
    VK_ACCESS_MEMORY_WRITE_BIT = 0x00010000,
    // Provided by VK_VERSION_1_3
      VK_ACCESS_NONE = 0,
    // Provided by VK_EXT_transform_feedback
      VK_ACCESS_TRANSFORM_FEEDBACK_WRITE_BIT_EXT = 0x02000000,
    // Provided by VK_EXT_transform_feedback
      VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT = 0x04000000,
    // Provided by VK_EXT_transform_feedback
      VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT = 0x08000000,
    // Provided by VK_EXT_conditional_rendering
      VK_ACCESS_CONDITIONAL_RENDERING_READ_BIT_EXT = 0x00100000,
    // Provided by VK_EXT_blend_operation_advanced
      VK_ACCESS_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT = 0x00080000,
    // Provided by VK_KHR_acceleration_structure
      VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR = 0x00200000,
    // Provided by VK_KHR_acceleration_structure
      VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR = 0x00400000,
    // Provided by VK_EXT_fragment_density_map
      VK_ACCESS_FRAGMENT_DENSITY_MAP_READ_BIT_EXT = 0x01000000,
    // Provided by VK_KHR_fragment_shading_rate
      VK_ACCESS_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR = 0x00800000,
    // Provided by VK_NV_device_generated_commands
      VK_ACCESS_COMMAND_PREPROCESS_READ_BIT_NV = 0x00020000,
    // Provided by VK_NV_device_generated_commands
      VK_ACCESS_COMMAND_PREPROCESS_WRITE_BIT_NV = 0x00040000,
    // Provided by VK_NV_shading_rate_image
      VK_ACCESS_SHADING_RATE_IMAGE_READ_BIT_NV = VK_ACCESS_FRAGMENT_SHADING_RATE_ATTACHMENT_READ_BIT_KHR,
    // Provided by VK_NV_ray_tracing
      VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_NV = VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR,
    // Provided by VK_NV_ray_tracing
      VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_NV = VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR,
    // Provided by VK_KHR_synchronization2
      VK_ACCESS_NONE_KHR = VK_ACCESS_NONE,
} VkAccessFlagBits;
```

```
VkSubpassDependency
srcSubpass; // index
dstSubpass; // index
srcStageMask;
dstStageMask;
srcAccessMask;
dstAccessMask;
dependencyFlags;
```

```
typedef enum VkDependencyFlagBits {
    VK_DEPENDENCY_BY_REGION_BIT = 0x00000001,
  // Provided by VK_VERSION_1_1
    VK_DEPENDENCY_DEVICE_GROUP_BIT = 0x00000004,
  // Provided by VK_VERSION_1_1
    VK_DEPENDENCY_VIEW_LOCAL_BIT = 0x00000002,
  // Provided by VK_EXT_attachment_feedback_loop_layout
    VK_DEPENDENCY_FEEDBACK_LOOP_BIT_EXT = 0x00000008,
  // Provided by VK_KHR_multiview
    VK_DEPENDENCY_VIEW_LOCAL_BIT_KHR = VK_DEPENDENCY_VIEW_LOCAL_BIT,
  // Provided by VK_KHR_device_group
    VK_DEPENDENCY_DEVICE_GROUP_BIT_KHR = VK_DEPENDENCY_DEVICE_GROUP_BIT,
} VkDependencyFlagBits;
```

VkAttachmentDescription for VkRenderPass

```
typedef struct VkAttachmentDescription {
    VkAttachmentDescriptionFlags flags;// usually 0
    VkFormat                     format;
    VkSampleCountFlagBits        samples;
    VkAttachmentLoadOp           loadOp;
    VkAttachmentStoreOp          storeOp;
    VkAttachmentLoadOp           stencilLoadOp;
    VkAttachmentStoreOp          stencilStoreOp;
    VkImageLayout                initialLayout;
    VkImageLayout                finalLayout;
} VkAttachmentDescription;
```

```
typedef enum VkFormat {
    ...
    VK_FORMAT_R32_UINT = 98,
    VK_FORMAT_R32_SINT = 99,
    VK_FORMAT_R32_SFLOAT = 100,
    VK_FORMAT_R32G32_UINT = 101,
    VK_FORMAT_R32G32_SINT = 102,
    VK_FORMAT_R32G32_SFLOAT = 103,
    VK_FORMAT_R32G32B32_UINT = 104,
    VK_FORMAT_R32G32B32_SINT = 105,
    VK_FORMAT_R32G32B32_SFLOAT = 106,
    VK_FORMAT_R32G32B32A32_UINT = 107,
    VK_FORMAT_R32G32B32A32_SINT = 108,
    VK_FORMAT_R32G32B32A32_SFLOAT = 109,
    ...
} VkFormat;
```

```
typedef enum VkSampleCountFlagBits {
    VK_SAMPLE_COUNT_1_BIT = 0x00000001,
    VK_SAMPLE_COUNT_2_BIT = 0x00000002,
    VK_SAMPLE_COUNT_4_BIT = 0x00000004,
    VK_SAMPLE_COUNT_8_BIT = 0x00000008,
    VK_SAMPLE_COUNT_16_BIT = 0x00000010,
    VK_SAMPLE_COUNT_32_BIT = 0x00000020,
    VK_SAMPLE_COUNT_64_BIT = 0x00000040,
} VkSampleCountFlagBits;
```

```
typedef enum VkAttachmentLoadOp {
    VK_ATTACHMENT_LOAD_OP_LOAD = 0,
    VK_ATTACHMENT_LOAD_OP_CLEAR = 1,
    VK_ATTACHMENT_LOAD_OP_DONT_CARE = 2,
  // Provided by VK_EXT_load_store_op_none
    VK_ATTACHMENT_LOAD_OP_NONE_EXT = 1000400000,
} VkAttachmentLoadOp;
```

```
typedef enum VkAttachmentStoreOp {
    VK_ATTACHMENT_STORE_OP_STORE = 0,
    VK_ATTACHMENT_STORE_OP_DONT_CARE = 1,
  // Provided by VK_VERSION_1_3
    VK_ATTACHMENT_STORE_OP_NONE = 1000301000,
  // Provided by VK_KHR_dynamic_rendering
    VK_ATTACHMENT_STORE_OP_NONE_KHR = VK_ATTACHMENT_STORE_OP_NONE,
  // Provided by VK_QCOM_render_pass_store_ops
    VK_ATTACHMENT_STORE_OP_NONE_QCOM = VK_ATTACHMENT_STORE_OP_NONE,
  // Provided by VK_EXT_load_store_op_none
    VK_ATTACHMENT_STORE_OP_NONE_EXT = VK_ATTACHMENT_STORE_OP_NONE,
} VkAttachmentStoreOp;
```

```
typedef enum VkImageLayout {
    VK_IMAGE_LAYOUT_UNDEFINED = 0,
    VK_IMAGE_LAYOUT_GENERAL = 1,
    VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL = 2,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL = 3,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL = 4,
    VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL = 5,
    VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL = 6,
    VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL = 7,
    VK_IMAGE_LAYOUT_PREINITIALIZED = 8,
    ...
} VkImageLayout;
```

VkRenderPass

```
VkRenderPassCreateInfo
sType = VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO;
pNext = nullptr;
flags = 0;
attachmentCount;
pAttachments;
subpassCount;
pSubpasses;
dependencyCount;
pDependencies;
```

VkAttachmentDescription for VkRenderPass

VkSubpassDescription for VkRenderPass

VkSubpassDependency for VkRenderPass

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

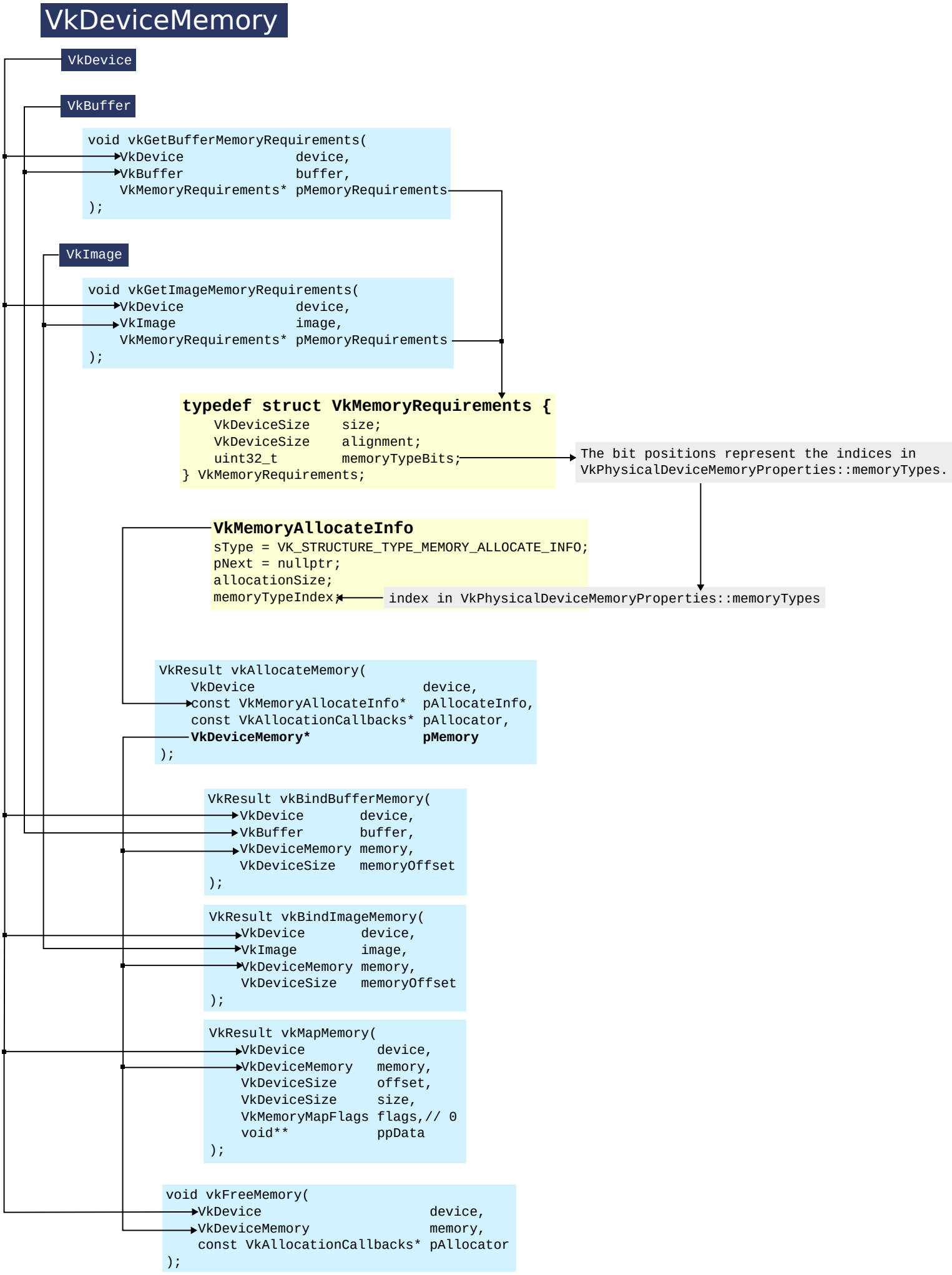
VkDevice

VkRenderPass

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

VkDevice

VkRenderPass


VkDeviceMemory
VkDevice
VkBuffer
void vkGetBufferMemoryRequirements(
VkDevice device,
VkBuffer buffer,
VkMemoryRequirements* pMemoryRequirements
);
VkImage
void vkGetImageMemoryRequirements(
VkDevice device,
VkImage image,
VkMemoryRequirements* pMemoryRequirements
);
typedef struct VkMemoryRequirements {
VkDeviceSize size;
VkDeviceSize alignment;
uint32_t memoryTypeBits;
} VkMemoryRequirements;
The bit positions represent the indices in VkPhysicalDeviceMemoryProperties::memoryTypes.
VkMemoryAllocateInfo
sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
pNext = nullptr;
allocationSize;
memoryTypeIndex;
index in VkPhysicalDeviceMemoryProperties::memoryTypes
VkResult vkAllocateMemory(
VkDevice device,
const VkMemoryAllocateInfo* pAllocateInfo,
const VkAllocationCallbacks* pAllocator,
VkDeviceMemory* pMemory
);
VkResult vkBindBufferMemory(
VkDevice device,
VkBuffer buffer,
VkDeviceMemory memory,
VkDeviceSize memoryOffset
);
VkResult vkBindImageMemory(
VkDevice device,
VkImage image,
VkDeviceMemory memory,
VkDeviceSize memoryOffset
);
VkResult vkMapMemory(
VkDevice device,
VkDeviceMemory memory,
VkDeviceSize offset,
VkDeviceSize size,
VkMemoryMapFlags flags,// 0
void** ppData
);
void vkFreeMemory(
VkDevice device,
VkDeviceMemory memory,
const VkAllocationCallbacks* pAllocator
);

VkBuffer

```
typedef enum VkBufferCreateFlagBits {
    VK_BUFFER_CREATE_SPARSE_BINDING_BIT = 0x00000001,
    VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT = 0x00000002,
    VK_BUFFER_CREATE_SPARSE_ALIASED_BIT = 0x00000004,
    VK_BUFFER_CREATE_PROTECTED_BIT = 0x00000008,
    VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT,
    VK_BUFFER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT,
    VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_EXT,
    VK_BUFFER_CREATE_DEVICE_ADDRESS_CAPTURE_REPLAY_BIT_KHR,
} VkBufferCreateFlagBits;
```

```
typedef enum VkBufferUsageFlagBits {
    VK_BUFFER_USAGE_TRANSFER_SRC_BIT = 0x00000001,
    VK_BUFFER_USAGE_TRANSFER_DST_BIT = 0x00000002,
    VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT = 0x00000004,
    VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT = 0x00000008,
    VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT = 0x00000010,
    VK_BUFFER_USAGE_STORAGE_BUFFER_BIT = 0x00000020,
    VK_BUFFER_USAGE_INDEX_BUFFER_BIT = 0x00000040,
    VK_BUFFER_USAGE_VERTEX_BUFFER_BIT = 0x00000080,
    VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT = 0x00000100,
...
} VkBufferUsageFlagBits;
```

```
typedef enum VkSharingMode {
    VK_SHARING_MODE_EXCLUSIVE = 0,
    VK_SHARING_MODE_CONCURRENT = 1,
} VkSharingMode;
```

```
VkBufferCreateInfo
sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
pNext = nullptr;
flags;
size;
usage;
sharingMode;
queueFamilyIndexCount;
pQueueFamilyIndices
```

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

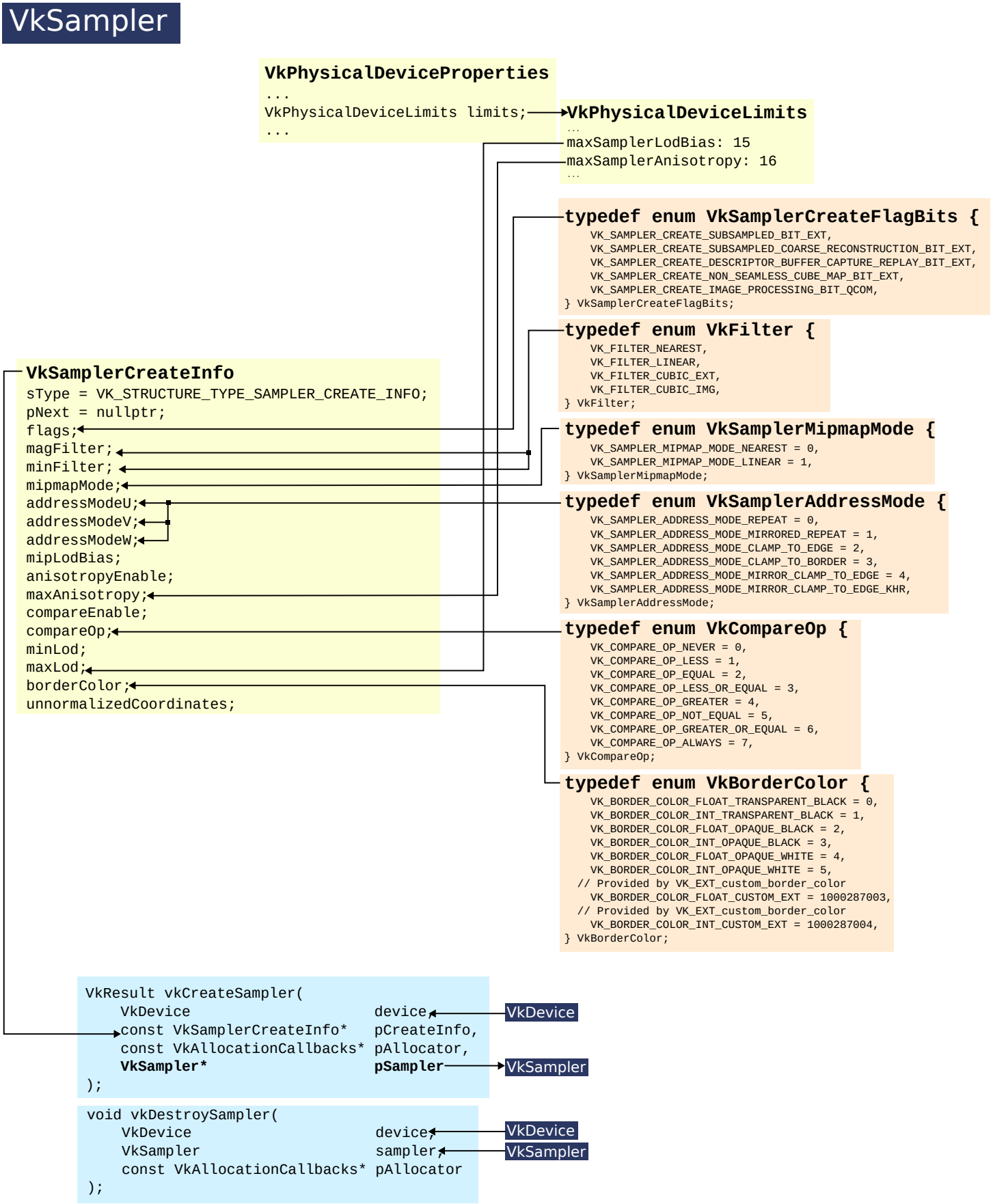
VkDevice

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

VkDevice

VkBuffer

VkSampler


VkPhysicalDeviceProperties
...
VkPhysicalDeviceLimits limits;
...
VkPhysicalDeviceLimits
...
maxSamplerLodBias: 15
maxSamplerAnisotropy: 16
...
typedef enum VkSamplerCreateFlagBits {
VK_SAMPLER_CREATE_SUBSAMPLED_BIT_EXT,
VK_SAMPLER_CREATE_SUBSAMPLED_COARSE_RECONSTRUCTION_BIT_EXT,
VK_SAMPLER_CREATE_DESCRIPTOR_BUFFER_CAPTURE_REPLAY_BIT_EXT,
VK_SAMPLER_CREATE_NON_SEAMLESS_CUBE_MAP_BIT_EXT,
VK_SAMPLER_CREATE_IMAGE_PROCESSING_BIT_QCOM,
} VkSamplerCreateFlagBits;
typedef enum VkFilter {
VK_FILTER_NEAREST,
VK_FILTER_LINEAR,
VK_FILTER_CUBIC_EXT,
VK_FILTER_CUBIC_IMG,
} VkFilter;
VkSamplerCreateInfo
sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO;
pNext = nullptr;
flags;
magFilter;
minFilter;
mipmapMode;
addressModeU;
addressModeV;
addressModeW;
mipLodBias;
anisotropyEnable;
maxAnisotropy;
compareEnable;
compareOp;
minLod;
maxLod;
borderColor;
unnormalizedCoordinates;
typedef enum VkSamplerMipmapMode {
VK_SAMPLER_MIPMAP_MODE_NEAREST = 0,
VK_SAMPLER_MIPMAP_MODE_LINEAR = 1,
} VkSamplerMipmapMode;
typedef enum VkSamplerAddressMode {
VK_SAMPLER_ADDRESS_MODE_REPEAT = 0,
VK_SAMPLER_ADDRESS_MODE_MIRRORED_REPEAT = 1,
VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE = 2,
VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER = 3,
VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE = 4,
VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE_KHR,
} VkSamplerAddressMode;
typedef enum VkCompareOp {
VK_COMPARE_OP_NEVER = 0,
VK_COMPARE_OP_LESS = 1,
VK_COMPARE_OP_EQUAL = 2,
VK_COMPARE_OP_LESS_OR_EQUAL = 3,
VK_COMPARE_OP_GREATER = 4,
VK_COMPARE_OP_NOT_EQUAL = 5,
VK_COMPARE_OP_GREATER_OR_EQUAL = 6,
VK_COMPARE_OP_ALWAYS = 7,
} VkCompareOp;
typedef enum VkBorderColor {
VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK = 0,
VK_BORDER_COLOR_INT_TRANSPARENT_BLACK = 1,
VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK = 2,
VK_BORDER_COLOR_INT_OPAQUE_BLACK = 3,
VK_BORDER_COLOR_FLOAT_OPAQUE_WHITE = 4,
VK_BORDER_COLOR_INT_OPAQUE_WHITE = 5,
// Provided by VK_EXT_custom_border_color
VK_BORDER_COLOR_FLOAT_CUSTOM_EXT = 1000287003,
// Provided by VK_EXT_custom_border_color
VK_BORDER_COLOR_INT_CUSTOM_EXT = 1000287004,
} VkBorderColor;
VkResult vkCreateSampler(
VkDevice device,
const VkSamplerCreateInfo* pCreateInfo,
const VkAllocationCallbacks* pAllocator,
VkSampler* pSampler
);
VkDevice
VkSampler
void vkDestroySampler(
VkDevice device,
VkSampler sampler,
const VkAllocationCallbacks* pAllocator
);
VkDevice
VkSampler

VkImage

```
VkResult vkGetSwapchainImagesKHR(
    VkDevice       device,
    VkSwapchainKHR swapchain,
    uint32_t*      pSwapchainImageCount,
    VkImage*       pSwapchainImages
);
```

VkDevice

VkSwapchainKHR

VkImage

```
typedef enum VkImageCreateFlagBits {
    VK_IMAGE_CREATE_SPARSE_BINDING_BIT = 0x00000001,
    VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT = 0x00000002,
    VK_IMAGE_CREATE_SPARSE_ALIASED_BIT = 0x00000004,
    VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT = 0x00000008,
    VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT = 0x00000010,
    ....
} VkImageCreateFlagBits;
```

```
typedef enum VkSampleCountFlagBits {
    VK_SAMPLE_COUNT_1_BIT = 0x00000001,
    VK_SAMPLE_COUNT_2_BIT = 0x00000002,
    VK_SAMPLE_COUNT_4_BIT = 0x00000004,
    VK_SAMPLE_COUNT_8_BIT = 0x00000008,
    VK_SAMPLE_COUNT_16_BIT = 0x00000010,
    VK_SAMPLE_COUNT_32_BIT = 0x00000020,
    VK_SAMPLE_COUNT_64_BIT = 0x00000040,
} VkSampleCountFlagBits;
```

```
typedef enum VkImageType {
    VK_IMAGE_TYPE_1D = 0,
    VK_IMAGE_TYPE_2D = 1,
    VK_IMAGE_TYPE_3D = 2,
} VkImageType;
```

```
typedef enum VkImageTiling {
    VK_IMAGE_TILING_OPTIMAL = 0,
    VK_IMAGE_TILING_LINEAR = 1,
  // Provided by VK_EXT_image_drm_format_modifier
    VK_IMAGE_TILING_DRM_FORMAT_MODIFIER_EXT = 1000158000,
} VkImageTiling;
```

```
typedef enum VkFormat {
    ...
    VK_FORMAT_R32_UINT = 98,
    VK_FORMAT_R32_SINT = 99,
    VK_FORMAT_R32_SFLOAT = 100,
    VK_FORMAT_R32G32_UINT = 101,
    VK_FORMAT_R32G32_SINT = 102,
    VK_FORMAT_R32G32_SFLOAT = 103,
    VK_FORMAT_R32G32B32_UINT = 104,
    VK_FORMAT_R32G32B32_SINT = 105,
    VK_FORMAT_R32G32B32_SFLOAT = 106,
    VK_FORMAT_R32G32B32A32_UINT = 107,
    VK_FORMAT_R32G32B32A32_SINT = 108,
    VK_FORMAT_R32G32B32A32_SFLOAT = 109,
    ...
} VkFormat;
```

```
typedef enum VkImageUsageFlagBits {
    VK_IMAGE_USAGE_TRANSFER_SRC_BIT = 0x00000001,
    VK_IMAGE_USAGE_TRANSFER_DST_BIT = 0x00000002,
    VK_IMAGE_USAGE_SAMPLED_BIT = 0x00000004,
    VK_IMAGE_USAGE_STORAGE_BIT = 0x00000008,
    VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT = 0x00000010,
    VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT = 0x00000020,
    VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT = 0x00000040,
    VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT = 0x00000080,
    ...
} VkImageUsageFlagBits;
```

```
typedef struct VkExtent3D {
    uint32_t    width;
    uint32_t    height;
    uint32_t    depth;
} VkExtent3D;
```

```
typedef enum VkSharingMode {
    VK_SHARING_MODE_EXCLUSIVE = 0,
    VK_SHARING_MODE_CONCURRENT = 1,
} VkSharingMode;
```

```
typedef enum VkImageLayout {
    VK_IMAGE_LAYOUT_UNDEFINED = 0,
    VK_IMAGE_LAYOUT_GENERAL = 1,
    VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL = 2,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL = 3,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL = 4,
    VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL = 5,
    VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL = 6,
    VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL = 7,
    VK_IMAGE_LAYOUT_PREINITIALIZED = 8,
    ...
} VkImageLayout;
```

```
VkImageCreateInfo
sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
pNext = nullptr;
flags;
imageType;
format;
extent;
mipLevels;// the number of levels
arrayLayers; // the number of layers
samples;
tiling;
usage;
sharingMode;
queueFamilyIndexCount;
pQueueFamilyIndices;
initialLayout;
```

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

VkDevice

VkImage

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

VkDevice

VkImage

VkImageView

```
VkImageViewCreateInfo
sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;;
pNext = // usually nullptr;
flags = // usually 0;
image;                                   VkImage
viewType // VK_IMAGE_VIEW_TYPE_2D;
format; // Ex. VK_FORMAT_B8G8R8A8_SRGB

// swizzling is like var.[xyzw] in shader language
components.r = VK_COMPONENT_SWIZZLE_IDENTITY;
    // VK_COMPONENT_SWIZZLE_R

components.g = VK_COMPONENT_SWIZZLE_IDENTITY;
    // VK_COMPONENT_SWIZZLE_G

components.b = VK_COMPONENT_SWIZZLE_IDENTITY;
    // VK_COMPONENT_SWIZZLE_B

components.a = VK_COMPONENT_SWIZZLE_IDENTITY;
    // VK_COMPONENT_SWIZZLE_A

subresourceRange.aspectMask =VK_IMAGE_ASPECT_COLOR_BIT;
subresourceRange.baseMipLevel = 0;
subresourceRange.levelCount = 1;
subresourceRange.baseArrayLayer = 1;
subresourceRange.layerCount = 1;
```

```
typedef enum VkFormat {
    ...
    VK_FORMAT_R8G8B8_UNORM = 23,
    ...
    VK_FORMAT_R8G8B8_SRGB = 29,
    ...
} VkFormat;
```

```
enum VkImageAspectFlagBits
VK_IMAGE_ASPECT_COLOR_BIT = 0x00000001,
VK_IMAGE_ASPECT_DEPTH_BIT = 0x00000002,
VK_IMAGE_ASPECT_STENCIL_BIT = 0x00000004,
VK_IMAGE_ASPECT_METADATA_BIT = 0x00000008,
VK_IMAGE_ASPECT_PLANE_0_BIT = 0x00000010,
VK_IMAGE_ASPECT_PLANE_1_BIT = 0x00000020,
VK_IMAGE_ASPECT_PLANE_2_BIT = 0x00000040,
VK_IMAGE_ASPECT_NONE = 0,
VK_IMAGE_ASPECT_MEMORY_PLANE_0_BIT_EXT = 0x00000080,
VK_IMAGE_ASPECT_MEMORY_PLANE_1_BIT_EXT = 0x00000100,
VK_IMAGE_ASPECT_MEMORY_PLANE_2_BIT_EXT = 0x00000200,
VK_IMAGE_ASPECT_MEMORY_PLANE_3_BIT_EXT = 0x00000400,
VK_IMAGE_ASPECT_PLANE_0_BIT_KHR = VK_IMAGE_ASPECT_PLANE_0_BIT,
VK_IMAGE_ASPECT_PLANE_1_BIT_KHR = VK_IMAGE_ASPECT_PLANE_1_BIT,
VK_IMAGE_ASPECT_PLANE_2_BIT_KHR = VK_IMAGE_ASPECT_PLANE_2_BIT,
VK_IMAGE_ASPECT_NONE_KHR = VK_IMAGE_ASPECT_NONE,
```

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

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

VkFramebuffer

```
typedef enum VkFramebufferCreateFlagBits {
  // Provided by VK_VERSION_1_2
    VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT = 0x00000001,
  // Provided by VK_KHR_imageless_framebuffer
    VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT_KHR
        = VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT,
} VkFramebufferCreateFlagBits;
```

VkFramebufferCreateInfo

```
sType = VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO;
pNext = nullptr;
flags;
renderPass;
attachmentCount;
pAttachments;
width;
height;
layers; // usually 1
```

VkRenderPass

VkImageView
must be aligned with
VkRenderPassCreateInfo::pAttachments,

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

VkDevice

VkFramebuffer

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

VkDevice

VkFramebuffer

VkCommandPool

```
typedef enum VkCommandPoolCreateFlagBits {
    VK_COMMAND_POOL_CREATE_TRANSIENT_BIT = 0x00000001,
    VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT = 0x00000002,
  // Provided by VK_VERSION_1_1
    VK_COMMAND_POOL_CREATE_PROTECTED_BIT = 0x00000004,
} VkCommandPoolCreateFlagBits;
```

```
VkCommandPoolCreateInfo
sType = VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO;
pNext = nullptr;
flags = VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;
queueFamilyIndex;
```

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

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

VkCommandBuffer

```
typedef enum VkCommandBufferLevel {
    VK_COMMAND_BUFFER_LEVEL_PRIMARY = 0,
    VK_COMMAND_BUFFER_LEVEL_SECONDARY = 1,
} VkCommandBufferLevel;
```

```
VkCommandBufferAllocateInfo
sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
pNext = nullptr;
commandPool;
level;
commandBufferCount; // num buffers to allocate
```

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

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

VkDescriptorSetLayout

```
typedef enum VkDescriptorType {
    VK_DESCRIPTOR_TYPE_SAMPLER = 0,
    VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER = 1,
    VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE = 2,
    VK_DESCRIPTOR_TYPE_STORAGE_IMAGE = 3,
    VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER = 4,
    VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER = 5,
    VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER = 6,
    VK_DESCRIPTOR_TYPE_STORAGE_BUFFER = 7,
    VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC = 8,
    VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC = 9,
    VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT = 10,
...
} VkDescriptorType;
```

```
typedef enum VkShaderStageFlagBits {
    VK_SHADER_STAGE_VERTEX_BIT = 0x00000001,
    VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT = 0x00000002,
    VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT = 0x00000004,
    VK_SHADER_STAGE_GEOMETRY_BIT = 0x00000008,
    VK_SHADER_STAGE_FRAGMENT_BIT = 0x00000010,
    VK_SHADER_STAGE_COMPUTE_BIT = 0x00000020,
    VK_SHADER_STAGE_ALL_GRAPHICS = 0x0000001F,
    VK_SHADER_STAGE_ALL = 0x7FFFFFFF,
...
} VkShaderStageFlagBits;
```

```
typedef struct VkDescriptorSetLayoutBinding {
    uint32_t              binding;
        // This must match the binding number in the shaders.
        // Ex. layout(binding = 0) uniform UniformBufferObject{...}ubo;
        //     layout(binding = 1) uniform sampler2D texSampler;

    VkDescriptorType      descriptorType;

    uint32_t              descriptorCount;
        // number of values in the array

    VkShaderStageFlags    stageFlags;

    const VkSampler*      pImmutableSamplers; // usually nullptr
} VkDescriptorSetLayoutBinding;
```

```
VkDescriptorSetLayoutCreateInfo {
sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
pNext = nullptr;
flags; // usually 0
bindingCount;
pBindings;
```

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

VkDevice

VkDescriptorSetLayout

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

VkDevice

VkDescriptorSetLayout

VkDescriptorPool

```
typedef enum VkDescriptorType {
    VK_DESCRIPTOR_TYPE_SAMPLER = 0,
    VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER = 1,
    VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE = 2,
    VK_DESCRIPTOR_TYPE_STORAGE_IMAGE = 3,
    VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER = 4,
    VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER = 5,
    VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER = 6,
    VK_DESCRIPTOR_TYPE_STORAGE_BUFFER = 7,
    VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC = 8,
    VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC = 9,
    VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT = 10,
...
} VkDescriptorType;
```

```
typedef struct VkDescriptorPoolSize {
    VkDescriptorType    type;
    uint32_t            descriptorCount;
} VkDescriptorPoolSize;
```

```
VkDescriptorPoolCreateInfo
sType = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
pNext = nullptr;
flags = 0;
maxSets;
poolSizeCount;
pPoolSizes;
```

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

VkDevice

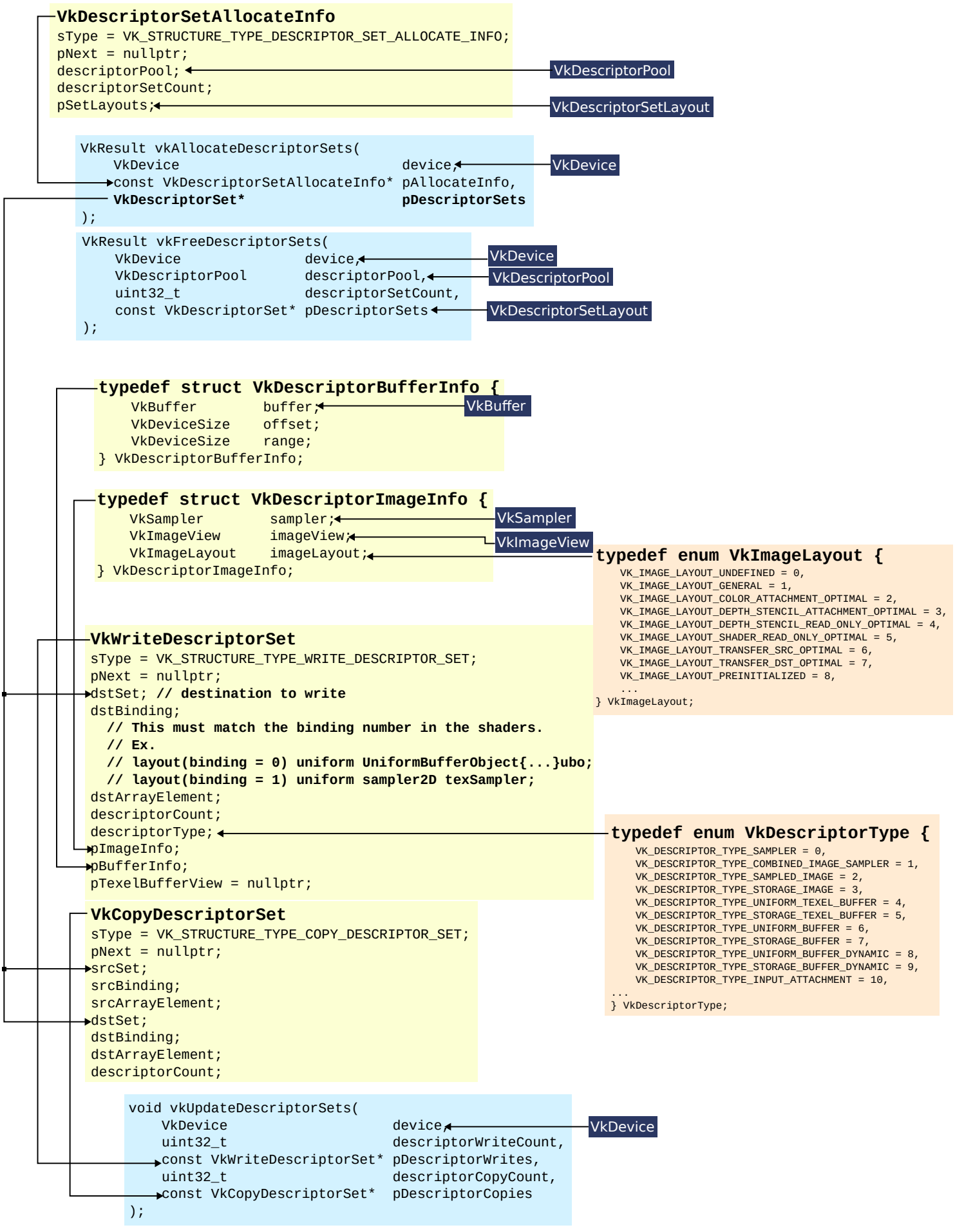
VkDescriptorPool

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

VkDevice

VkDescriptorPool

VkDescriptorSet



VkSemaphore

```
VkSemaphoreCreateInfo
sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
pNext = nullptr;
flags = 0;
```

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

VkDevice

VkSemaphore

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

VkDevice

VkSemaphore

VkFence

```
typedef enum VkFenceCreateFlagBits {
    VK_FENCE_CREATE_SIGNALED_BIT = 0x00000001,
} VkFenceCreateFlagBits;
```

```
VkFenceCreateInfo
sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
pNext = nullptr;
flags;
```

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

VkDevice

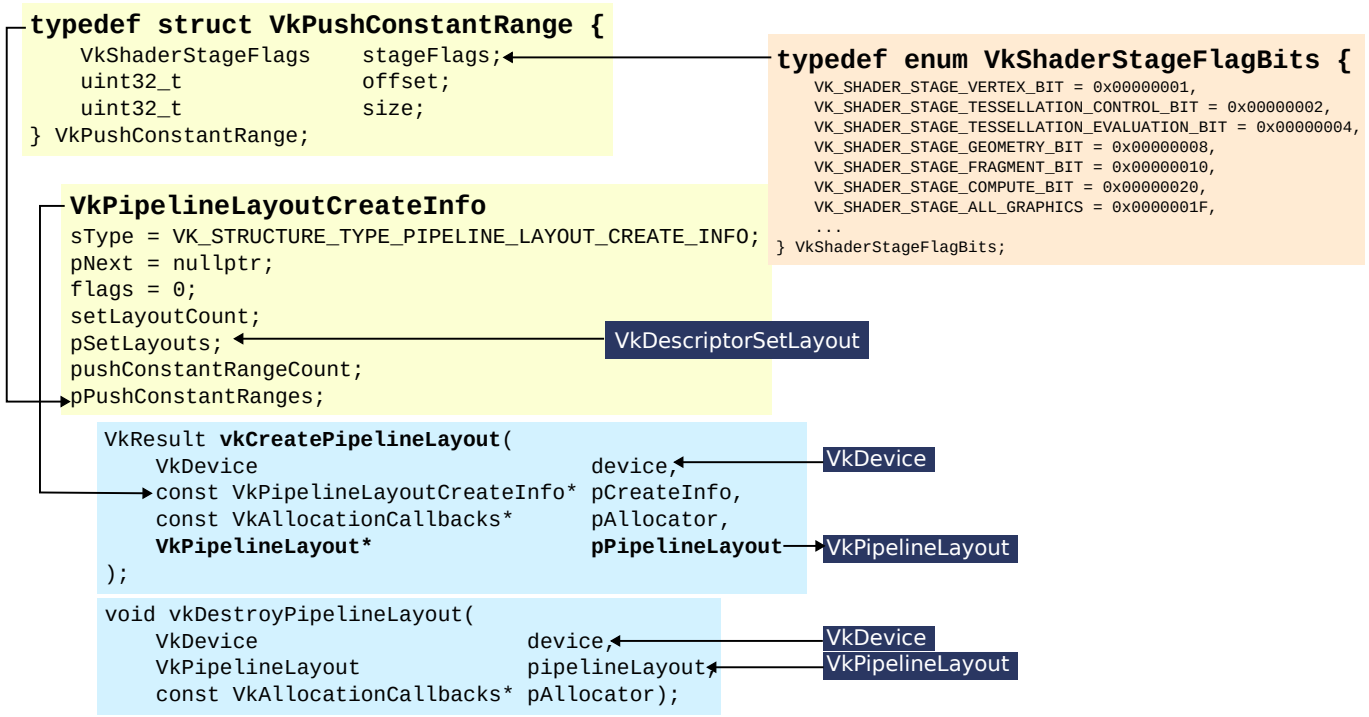
VkFence

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

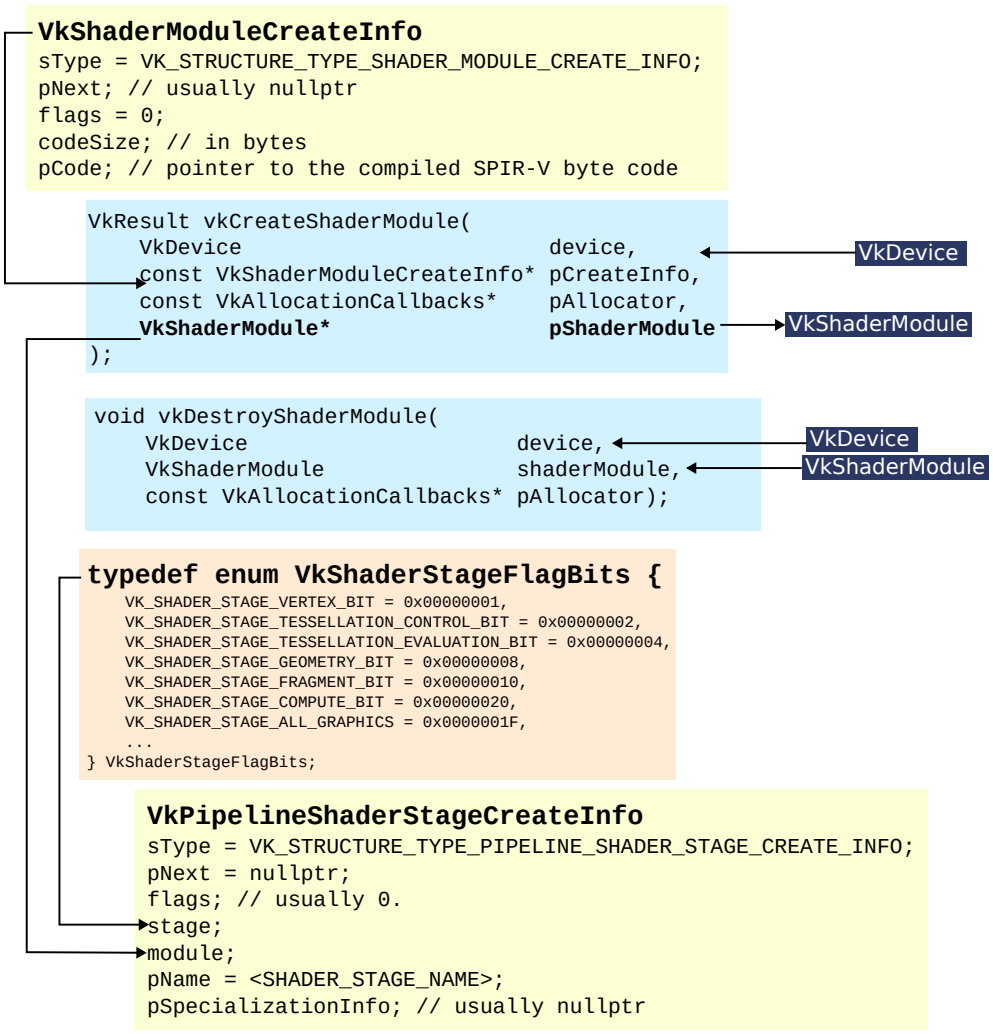
VkDevice

VkFence

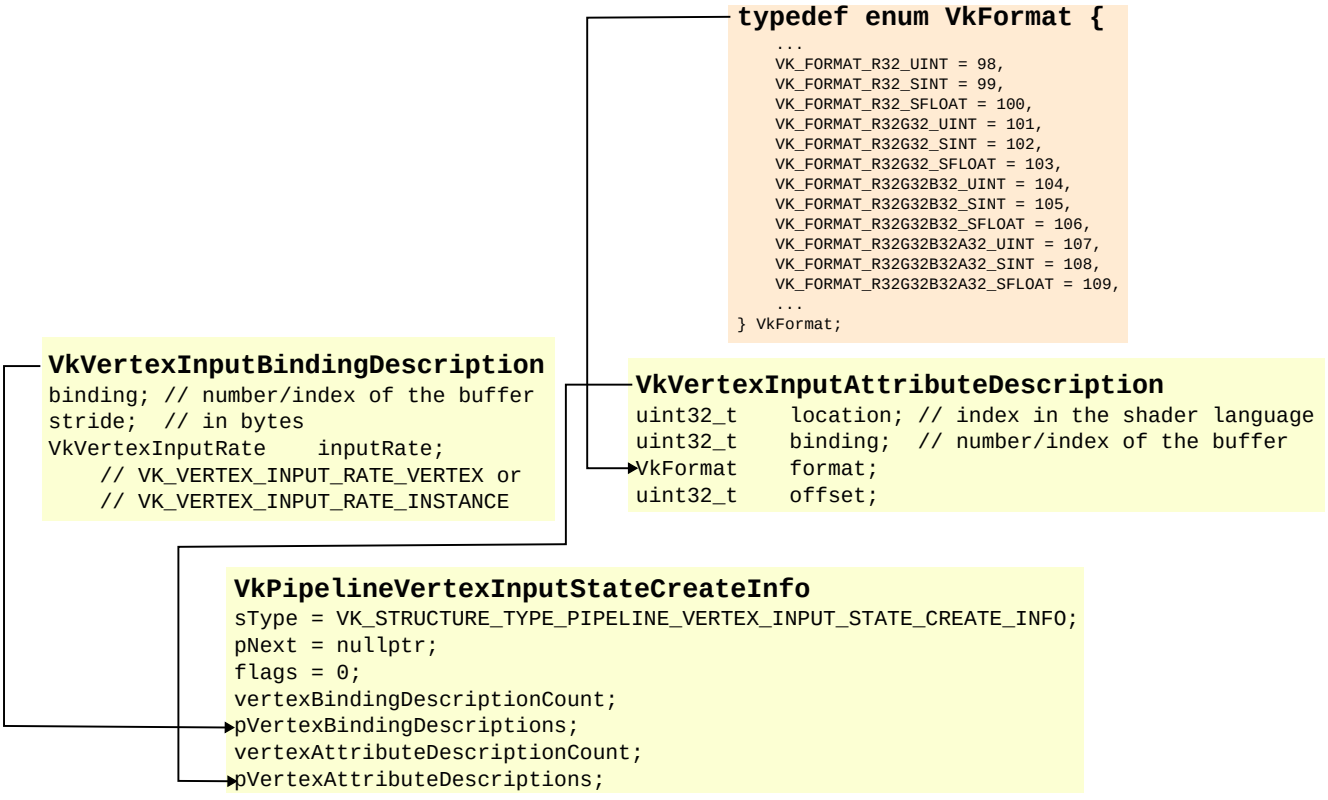
VkPipelineLayout for VkGraphicsPipelineCreateInfo & vkCmdBindDescriptorSets()



VkPipelineShaderStageCreateInfo for VkGraphicsPipelineCreateInfo



VkPipelineVertexInputStateCreateInfo for VkGraphicsPipelineCreateInfo



VkPipelineInputAssemblyStateCreateInfo for VkGraphicsPipelineCreateInfo

```
typedef enum VkPrimitiveTopology {
    VK_PRIMITIVE_TOPOLOGY_POINT_LIST = 0,
    VK_PRIMITIVE_TOPOLOGY_LINE_LIST = 1,
    VK_PRIMITIVE_TOPOLOGY_LINE_STRIP = 2,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST = 3,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP = 4,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN = 5,
    VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY = 6,
    VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY = 7,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY = 8,
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY = 9,
    VK_PRIMITIVE_TOPOLOGY_PATCH_LIST = 10,
} VkPrimitiveTopology;
```

```
VkPipelineInputAssemblyStateCreateInfo
sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
pNext = nullptr;
flags = 0;
topology;
primitiveRestartEnable = VK_FALSE; // used for triangle strips etc.
```

VkPipelineTessellationStateCreateInfo for VkGraphicsPipelineCreateInfo

```
VkPipelineTessellationStateCreateInfo
sType = VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_STATE_CREATE_INFO;
pNext = nullptr;
flags = 0;
patchControlPoints; // number of control points per patch
```

VkPipelineViewportStateCreateInfo for VkGraphicsPipelineCreateInfo

```
// Y-down coordinate system.
typedef struct VkViewport {
    float    x;// left
    float    y;// top
    float    width;
    float    height;
    float    minDepth = 0.0;
    float    maxDepth = 1.0;
} VkViewport;
```

```
VkPipelineViewportStateCreateInfo
sType = VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO;
pNext = nullptr;
flags = 0;
viewportCount = 1;
pViewports;
scissorCount = 1;
const VkRect2D* pScissors;
```

VkPipelineRasterizationStateCreateInfo for VkGraphicsPipelineCreateInfo

```
VkPipelineRasterizationStateCreateInfo
sType = VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO;
pNext = nullptr;
flags = 0;
depthClampEnable; // usually VK_FALSE, for shadow mapping VK_TRUE
rasterizerDiscardEnable = VK_FALSE;
polygonMode = VK_POLYGON_MODE_FILL, VK_POLYGON_MODE_LINE, or VK_POLYGON_MODE_POINT.
cullMode VK_CULL_MODE_BACK_BIT;
frontFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
depthBiasEnable = VK_FALSE;
depthBiasConstantFactor = 0.0;
depthBiasClamp = 0.0;
depthBiasSlopeFactor = 0.0;
lineWidth = 1.0;
```

VkPipelineMultisampleStateCreateInfo for VkGraphicsPipelineCreateInfo

```
VkPipelineMultisampleStateCreateInfo
sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO;
pNext = nullptr;
flags = 0;
rasterizationSamples = VK_SAMPLE_COUNT_8_BIT;
sampleShadingEnable = VK_FALSE;
minSampleShading = 1.0; // don't care
pSampleMask = nullptr;
alphaToCoverageEnable = VK_FALSE;
alphaToOneEnable = VK_FALSE;
```

```
VkPhysicalDeviceLimits
...
framebufferColorSampleCounts: 0xf
framebufferDepthSampleCounts: 0xf
...
```

VkPipelineDepthStencilStateCreateInfo for VkGraphicsPipelineCreateInfo

```
typedef enum VkStencilOp {
    VK_STENCIL_OP_KEEP = 0,
    VK_STENCIL_OP_ZERO = 1,
    VK_STENCIL_OP_REPLACE = 2,
    VK_STENCIL_OP_INCREMENT_AND_CLAMP = 3,
    VK_STENCIL_OP_DECREMENT_AND_CLAMP = 4,
    VK_STENCIL_OP_INVERT = 5,
    VK_STENCIL_OP_INCREMENT_AND_WRAP = 6,
    VK_STENCIL_OP_DECREMENT_AND_WRAP = 7,
} VkStencilOp;
```

```
typedef enum VkCompareOp {
    VK_COMPARE_OP_NEVER = 0,
    VK_COMPARE_OP_LESS = 1,
    VK_COMPARE_OP_EQUAL = 2,
    VK_COMPARE_OP_LESS_OR_EQUAL = 3,
    VK_COMPARE_OP_GREATER = 4,
    VK_COMPARE_OP_NOT_EQUAL = 5,
    VK_COMPARE_OP_GREATER_OR_EQUAL = 6,
    VK_COMPARE_OP_ALWAYS = 7,
} VkCompareOp;
```

```
typedef struct VkStencilOpState {
    VkStencilOp    failOp;
    VkStencilOp    passOp;
    VkStencilOp    depthFailOp;
    VkCompareOp    compareOp;
    uint32_t       compareMask;
    uint32_t       writeMask;
    uint32_t       reference;
} VkStencilOpState;
```

```
VkPipelineDepthStencilStateCreateInfo
sType = VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO;
pNext = nullptr;
flags = 0;
depthTestEnable = VK_TRUE;
depthWriteEnable = VK_TRUE;
depthCompareOp = VK_COMPARE_OP_LESS;
depthBoundsTestEnable =VK_FALSE;
stencilTestEnable = VK_FALSE;
front;
back;
minDepthBounds = 0.0f;
maxDepthBounds = 1.0f;
```

VkPipelineColorBlendStateCreateInfo for VkGraphicsPipelineCreateInfo

```
typedef enum VkBlendFactor {
    VK_BLEND_FACTOR_ZERO = 0,
    VK_BLEND_FACTOR_ONE = 1,
    VK_BLEND_FACTOR_SRC_COLOR = 2,
    VK_BLEND_FACTOR_ONE_MINUS_SRC_COLOR = 3,
    VK_BLEND_FACTOR_DST_COLOR = 4,
    VK_BLEND_FACTOR_ONE_MINUS_DST_COLOR = 5,
    VK_BLEND_FACTOR_SRC_ALPHA = 6,
    VK_BLEND_FACTOR_ONE_MINUS_SRC_ALPHA = 7,
    VK_BLEND_FACTOR_DST_ALPHA = 8,
    VK_BLEND_FACTOR_ONE_MINUS_DST_ALPHA = 9,
    VK_BLEND_FACTOR_CONSTANT_COLOR = 10,
    VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_COLOR = 11,
    VK_BLEND_FACTOR_CONSTANT_ALPHA = 12,
    VK_BLEND_FACTOR_ONE_MINUS_CONSTANT_ALPHA = 13,
    VK_BLEND_FACTOR_SRC_ALPHA_SATURATE = 14,
    VK_BLEND_FACTOR_SRC1_COLOR = 15,
    VK_BLEND_FACTOR_ONE_MINUS_SRC1_COLOR = 16,
    VK_BLEND_FACTOR_SRC1_ALPHA = 17,
    VK_BLEND_FACTOR_ONE_MINUS_SRC1_ALPHA = 18,
} VkBlendFactor;
```

```
typedef enum VkBlendOp {
    VK_BLEND_OP_ADD = 0,
    VK_BLEND_OP_SUBTRACT = 1,
    VK_BLEND_OP_REVERSE_SUBTRACT = 2,
    VK_BLEND_OP_MIN = 3,
    VK_BLEND_OP_MAX = 4,
...
} VkBlendOp;
```

```
VkPipelineColorBlendAttachmentState
blendEnable = VK_FALSE/VK_TRUE;
srcColorBlendFactor;
dstColorBlendFactor;
colorBlendOp;
srcAlphaBlendFactor;
dstAlphaBlendFactor;
alphaBlendOp;
colorWriteMask = VK_COLOR_COMPONENT_R_BIT
               | VK_COLOR_COMPONENT_G_BIT
               | VK_COLOR_COMPONENT_B_BIT
               | VK_COLOR_COMPONENT_A_BIT;
```

```
typedef enum VkLogicOp {
    VK_LOGIC_OP_CLEAR = 0,
    VK_LOGIC_OP_AND = 1,
    VK_LOGIC_OP_AND_REVERSE = 2,
    VK_LOGIC_OP_COPY = 3,
    VK_LOGIC_OP_AND_INVERTED = 4,
    VK_LOGIC_OP_NO_OP = 5,
    VK_LOGIC_OP_XOR = 6,
    VK_LOGIC_OP_OR = 7,
    VK_LOGIC_OP_NOR = 8,
    VK_LOGIC_OP_EQUIVALENT = 9,
    VK_LOGIC_OP_INVERT = 10,
    VK_LOGIC_OP_OR_REVERSE = 11,
    VK_LOGIC_OP_COPY_INVERTED = 12,
    VK_LOGIC_OP_OR_INVERTED = 13,
    VK_LOGIC_OP_NAND = 14,
    VK_LOGIC_OP_SET = 15,
} VkLogicOp;
```

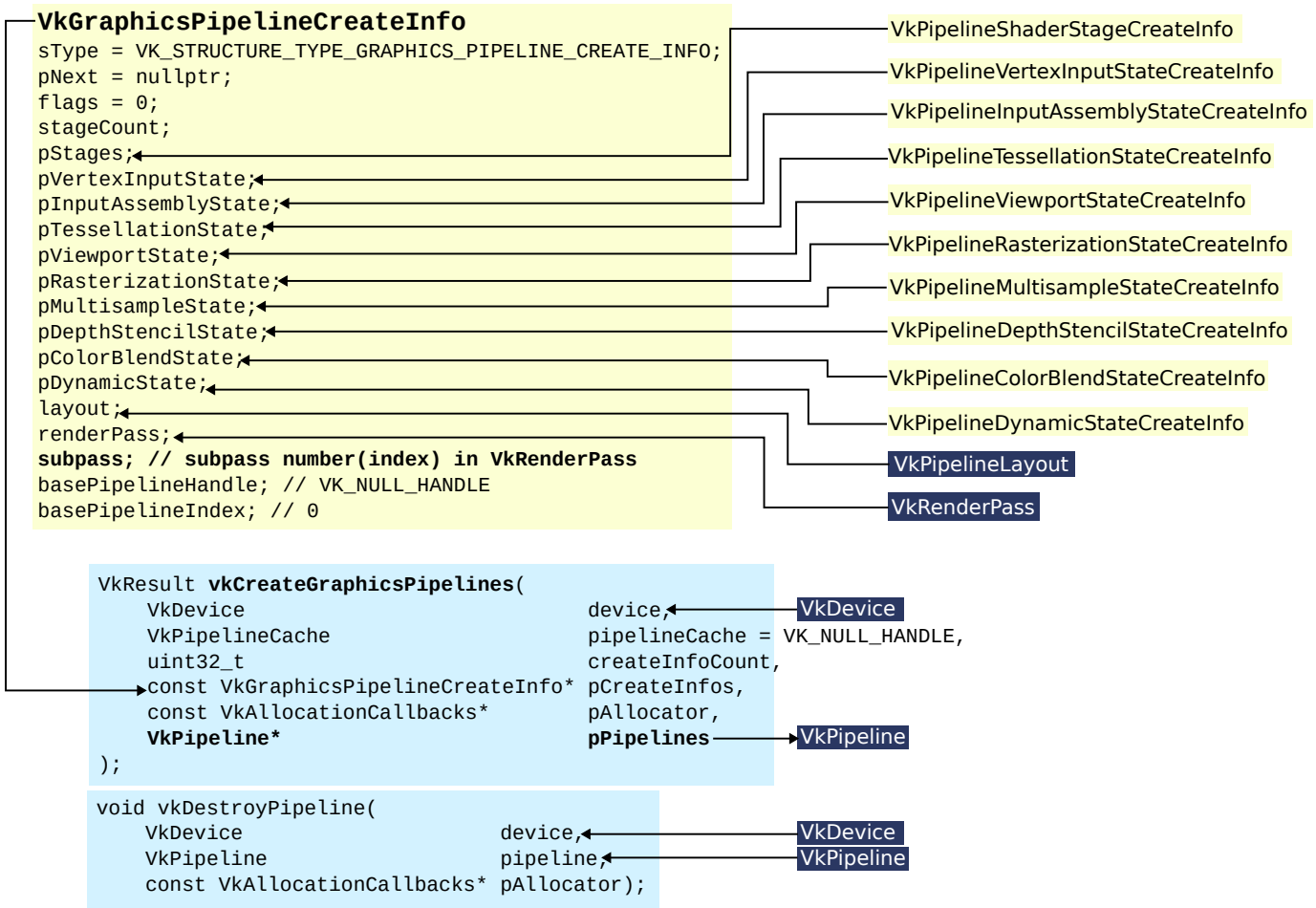
```
VkPipelineColorBlendStateCreateInfo
sType = VK_LOGIC_OP_COPY;
pNext = nullptr;
flags; // usually 0
logicOpEnable; // VK_FALSE
logicOp; // VK_LOGIC_OP_COPY
attachmentCount;
pAttachments;
blendConstants[4]; // = 0.0f for all;
} VkPipelineColorBlendStateCreateInfo;
```

VkPipelineDynamicStateCreateInfo for VkGraphicsPipelineCreateInfo

```
typedef enum VkDynamicState {
    VK_DYNAMIC_STATE_VIEWPORT = 0,
    VK_DYNAMIC_STATE_SCISSOR = 1,
    VK_DYNAMIC_STATE_LINE_WIDTH = 2,
    VK_DYNAMIC_STATE_DEPTH_BIAS = 3,
    VK_DYNAMIC_STATE_BLEND_CONSTANTS = 4,
    VK_DYNAMIC_STATE_DEPTH_BOUNDS = 5,
    VK_DYNAMIC_STATE_STENCIL_COMPARE_MASK = 6,
    VK_DYNAMIC_STATE_STENCIL_WRITE_MASK = 7,
...
} VkDynamicState;
```

```
VkPipelineDynamicStateCreateInfo
sType = VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO;
pNext = nullptr;
flags = 0;
dynamicStateCount;
pDynamicStates;
```

VkPipeline


VkGraphicsPipelineCreateInfo
sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
pNext = nullptr;
flags = 0;
stageCount;
pStages;
pVertexInputState;
pInputAssemblyState;
pTessellationState;
pViewportState;
pRasterizationState;
pMultisampleState;
pDepthStencilState;
pColorBlendState;
pDynamicState;
layout;
renderPass;
subpass; // subpass number(index) in VkRenderPass
basePipelineHandle; // VK_NULL_HANDLE
basePipelineIndex; // 0
VkPipelineShaderStageCreateInfo
VkPipelineVertexInputStateCreateInfo
VkPipelineInputAssemblyStateCreateInfo
VkPipelineTessellationStateCreateInfo
VkPipelineViewportStateCreateInfo
VkPipelineRasterizationStateCreateInfo
VkPipelineMultisampleStateCreateInfo
VkPipelineDepthStencilStateCreateInfo
VkPipelineColorBlendStateCreateInfo
VkPipelineDynamicStateCreateInfo
VkPipelineLayout
VkRenderPass
VkResult vkCreateGraphicsPipelines(
VkDevice device,
VkPipelineCache pipelineCache = VK_NULL_HANDLE,
uint32_t createInfoCount,
const VkGraphicsPipelineCreateInfo* pCreateInfos,
const VkAllocationCallbacks* pAllocator,
VkPipeline* pPipelines
);
VkDevice
VkPipeline
void vkDestroyPipeline(
VkDevice device,
VkPipeline pipeline,
const VkAllocationCallbacks* pAllocator);
VkDevice
VkPipeline

<OUTERMOST LOOP PER DRAW>

```
VkResult vkAcquireNextImageKHR(
    VkDevice       device,
    VkSwapchainKHR swapchain,
    uint64_t       timeout,
    VkSemaphore    semaphore,
    VkFence        fence,
    uint32_t*      pImageIndex
);
```

VkDevice

VkSwapchainKHR

VkSemaphore

VkFence

```
VkResult vkResetFences(
    VkDevice       device,
    uint32_t       fenceCount,
    const VkFence* pFences
);
```

VkDevice

VkFence

```
        <MID LOOP PER COMMAND BUFFER>
```

```
typedef enum VkPipelineStageFlagBits {
    VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT = 0x00000001,
    VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT = 0x00000002,
    VK_PIPELINE_STAGE_VERTEX_INPUT_BIT = 0x00000004,
    VK_PIPELINE_STAGE_VERTEX_SHADER_BIT = 0x00000008,
    VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT = 0x00000010,
    VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT = 0x00000020,
    VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT = 0x00000040,
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT = 0x00000080,
    VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT = 0x00000100,
    VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT = 0x00000200,
    VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT = 0x00000400,
    VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT = 0x00000800,
    VK_PIPELINE_STAGE_TRANSFER_BIT = 0x00001000,
    VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT = 0x00002000,
    VK_PIPELINE_STAGE_HOST_BIT = 0x00004000,
    VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT = 0x00008000,
    VK_PIPELINE_STAGE_ALL_COMMANDS_BIT = 0x00010000,
    // Provided by VK_VERSION_1_3
      VK_PIPELINE_STAGE_NONE = 0,
    // Provided by VK_EXT_transform_feedback
      VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT = 0x01000000,
    // Provided by VK_EXT_conditional_rendering
      VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EXT = 0x00040000,
    // Provided by VK_KHR_acceleration_structure
      VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR = 0x02000000,
    // Provided by VK_KHR_ray_tracing_pipeline
      VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR = 0x00200000,
    // Provided by VK_EXT_fragment_density_map
      VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BIT_EXT = 0x00800000,
    // Provided by VK_KHR_fragment_shading_rate
      VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR = 0x00400000,
    // Provided by VK_NV_device_generated_commands
      VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV = 0x00020000,
    // Provided by VK_EXT_mesh_shader
      VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT = 0x00080000,
    // Provided by VK_EXT_mesh_shader
      VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT = 0x00100000,
    // Provided by VK_NV_shading_rate_image
      VK_PIPELINE_STAGE_SHADING_RATE_IMAGE_BIT_NV = VK_PIPELINE_STAGE_FRAGMENT_SHADING_RATE_ATTACHMENT_BIT_KHR,
    // Provided by VK_NV_ray_tracing
      VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_NV = VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR,
    // Provided by VK_NV_ray_tracing
      VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_NV = VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUILD_BIT_KHR,
    // Provided by VK_NV_mesh_shader
      VK_PIPELINE_STAGE_TASK_SHADER_BIT_NV = VK_PIPELINE_STAGE_TASK_SHADER_BIT_EXT,
    // Provided by VK_NV_mesh_shader
      VK_PIPELINE_STAGE_MESH_SHADER_BIT_NV = VK_PIPELINE_STAGE_MESH_SHADER_BIT_EXT,
    // Provided by VK_KHR_synchronization2
      VK_PIPELINE_STAGE_NONE_KHR = VK_PIPELINE_STAGE_NONE,
} VkPipelineStageFlagBits;
```

```
VkSubmitInfo
sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
pNext = nullptr;
waitSemaphoreCount;
pWaitSemaphores;
pWaitDstStageMask;
commandBufferCount;
pCommandBuffers;
signalSemaphoreCount;
pSignalSemaphores;
```

VkSemaphore

VkCommandBuffers

```
VkResult vkQueueSubmit(
    VkQueue             queue,
    uint32_t            submitCount,
    const VkSubmitInfo* pSubmits,
    VkFence             fence
);
```

VkQueue

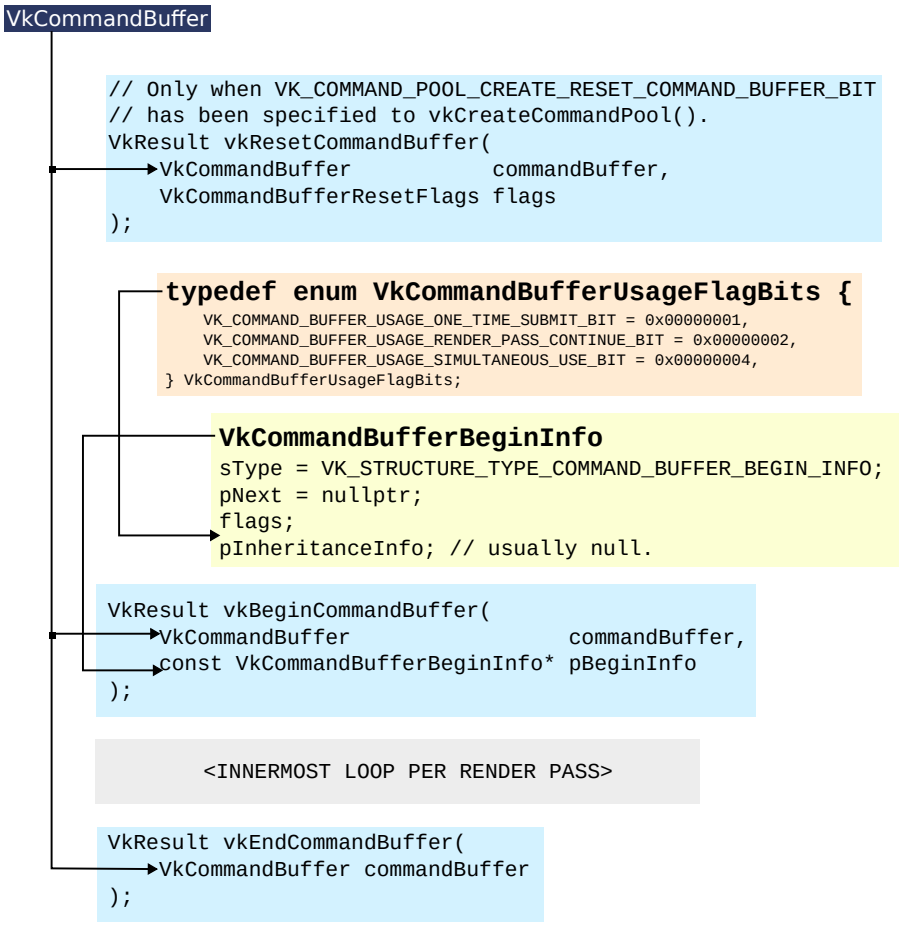
VkFence

```
VkPresentInfoKHR
sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
pNext = nullptr;
waitSemaphoreCount;
pWaitSemaphores;
swapchainCount;
pSwapchains;
pImageIndices;
pResults = nullptr;
```

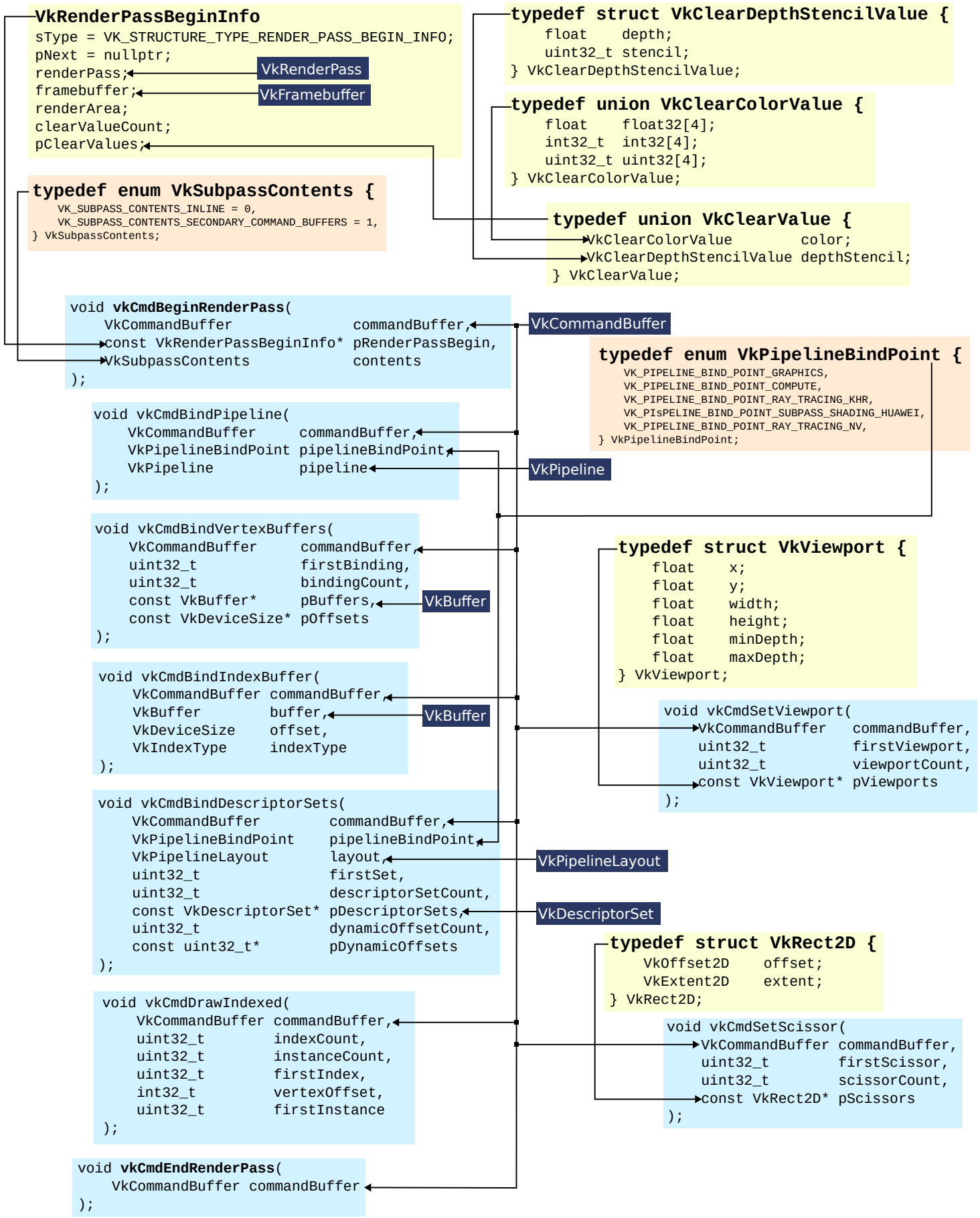
```
VkResult vkQueuePresentKHR(
    VkQueue                 queue,
    const VkPresentInfoKHR* pPresentInfo
);
```

VkQueue

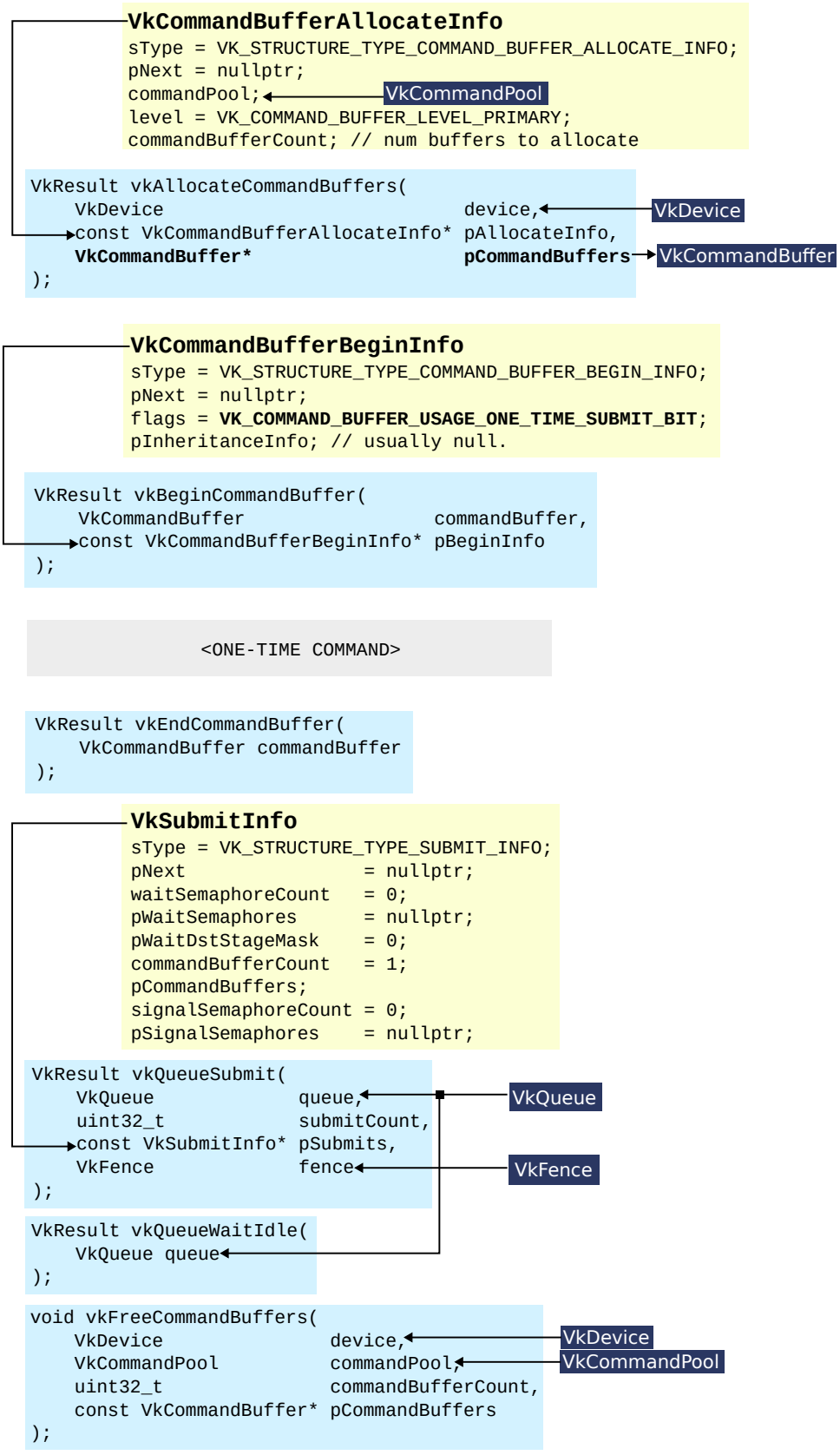
<MID LOOP PER COMMAND BUFFER>



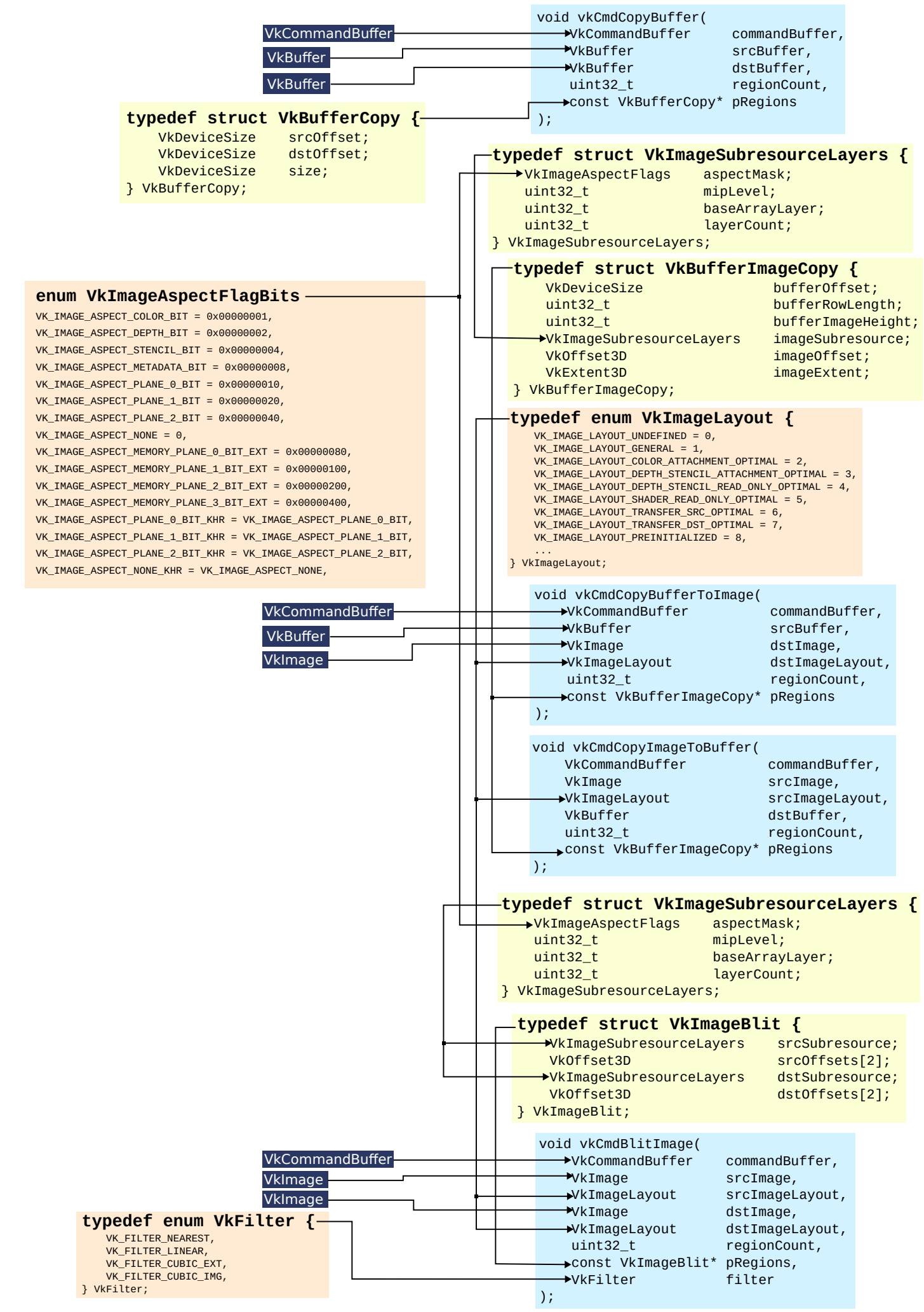
<INNERMOST LOOP PER RENDER PASS>



VkCommandBuffer for a One-Time Command



One-Time Command for Copying



Memory Barrier