Vulkan® is a graphics and compute API consisting of procedures and functions to specify shader programs, compute kernels, objects, and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects. Vulkan is also a pipeline with programmable and state-driven fixed-function stages that are invoked by a set of specific drawing operations.

Specification and additional resources at **www.khronos.org/vulkan**

Color coded names as follows: function names and structure names

[n.n.n] Indicates sections and text in the Vulkan API 1.1 Specification.

P.# Indicates a page in this reference guide for more information.

= 0 Indicates reserved for future use.

pNext must either be NULL, or point to a valid structure which extends the base structure according to the valid usage rules of the base structure.

Return Codes [2.7.3]

Return codes are reported via VkResult return values.

Success Codes

Success codes are non-negative.

VK_SUCCESS	VK_NOT_READY
VK_TIMEOUT	VK_EVENT_{SET, RESET}
VK_INCOMPLETE	VK_SUBOPTIMAL_KHR

Error Codes

Error codes are negative.

VK_ERROR_OUT_OF_{HOST, DEVICE}_MEMORY
VK_ERROR_{INITIALIZATION, MEMORY_MAP}_FAILED
VK_ERROR_DEVICE_LOST
VK_ERROR_{EXTENSION, FEATURE, LAYER}_NOT_PRESENT
VK_ERROR_INCOMPATIBLE_DRIVER
VK_ERROR_TOO_MANY_OBJECTS
VK_ERROR_FORMAT_NOT_SUPPORTED
VK_ERROR_FRAGMENTED_POOL
VK_ERROR_OUT_OF_POOL_MEMORY
VK_ERROR_INVALID_EXTERNAL_HANDLE
VK_ERROR_SURFACE_LOST_KHR
VK_ERROR_NATIVE_WINDOW_IN_USE_KHR
VK_ERROR_OUT_OF_DATE_KHR
VK_ERROR_INCOMPATIBLE_DISPLAY_KHR

Command Function Pointers and Instances [3]

Command Function Pointers [3.1]

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

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

PFN_vkVoidFunction is:

```
typedef void(VKAPI_PTR* PFN_vkVoidFunction)(void);
```

Instances [3.2]

```
VkResult vkEnumerateInstanceVersion(
    uint32_t* pApiVersion);

VkResult vkCreateInstance(
    const VkInstanceCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkInstance* pInstance);

typedef struct VkInstanceCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkInstanceCreateFlags flags; = 0
    const VkApplicationInfo* pApplicationInfo;
    uint32_t enabledLayerCount;
    const char* const* ppEnabledLayerNames;
    uint32_t enabledExtensionCount;
    const char* const* ppEnabledExtensionNames;
} VkInstanceCreateInfo;

typedef struct VkApplicationInfo {
    VkStructureType sType; P.15
    const void* pNext;
    const char* pApplicationName;
    uint32_t applicationVersion;
    const char* pEngineName;
    uint32_t engineVersion;
    uint32_t apiVersion;
} VkApplicationInfo;

void vkDestroyInstance(
    VkInstance instance,
    const VkAllocationCallbacks* pAllocator); P.12
```

Queues [4.3]

```
typedef struct VkDeviceQueueCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkDeviceQueueCreateFlags flags;
    uint32_t queueFamilyIndex;
    uint32_t queueCount;
    const float* pQueuePriorities;
} VkDeviceQueueCreateInfo;
```

flags: VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT

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

void vkGetDeviceQueue2(VkDevice device,
    const VkDeviceQueueInfo2* pQueueInfo,
    VkQueue* pQueue);

typedef struct VkDeviceQueueInfo2 {
    VkStructureType sType; P.15
    const void* pNext;
    VkDeviceQueueCreateFlags flags;
    uint32_t queueFamilyIndex; uint32_t queueIndex;
} VkDeviceQueueInfo2;
```

flags: VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT

Devices and Queues [4]

Physical Devices [4.1]

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

void vkGetPhysicalDeviceProperties(
    VkPhysicalDevice physicalDevice,
    VkPhysicalDeviceProperties* pProperties); P.14

void vkGetPhysicalDeviceProperties2(
    VkPhysicalDevice physicalDevice,
    VkPhysicalDeviceProperties2* pProperties);

typedef struct VkPhysicalDeviceProperties2 {
    VkStructureType sType; P.15
    void* pNext;
    VkPhysicalDeviceProperties properties; P.14
} VkPhysicalDeviceProperties2;
```

pNext must be NULL or point to one of:

VkPhysicalDeviceIDProperties P.14
VkPhysicalDeviceMaintenance3Properties P.14
VkPhysicalDeviceMultiviewProperties P.14
VkPhysicalDevicePointClippingProperties P.14
VkPhysicalDeviceProtectedMemoryProperties P.15
VkPhysicalDeviceSubgroupProperties P.15

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

void vkGetPhysicalDeviceQueueFamilyProperties2(
    VkPhysicalDevice physicalDevice,
    uint32_t* pQueueFamilyPropertyCount,
    VkQueueFamilyProperties2*pQueueFamilyProperties);

typedef struct VkQueueFamilyProperties {
    VkQueueFlags queueFlags;
    uint32_t queueCount;
    uint32_t timestampValidBits;
    VkExtent3D minImageTransferGranularity; P.12
} VkQueueFamilyProperties;
```

queueFlags:
VK_QUEUE_*X*_BIT where *X* is GRAPHICS, COMPUTE, TRANSFER, PROTECTED, SPARSE_BINDING

```
typedef struct VkQueueFamilyProperties2 {
    VkStructureType sType; P.15
    void* pNext; VkQueueFamilyProperties
        queueFamilyProperties;
} VkQueueFamilyProperties2;
```

Devices [4.2]

```
VkResult vkEnumeratePhysicalDeviceGroups(
    VkInstance instance,
    uint32_t* pPhysicalDeviceGroupCount,
    VkPhysicalDeviceGroupProperties*
        pPhysicalDeviceGroupProperties);

typedef struct VkPhysicalDeviceGroupProperties {
    VkStructureType sType; P.15
    void* pNext;
    uint32_t physicalDeviceCount;
    VkPhysicalDevice physicalDevices[
        VK_MAX_DEVICE_GROUP_SIZE];
    VkBool32 subsetAllocation;
} VkPhysicalDeviceGroupProperties;
```

Device Creation [4.2.1]

```
VkResult vkCreateDevice(
    VkPhysicalDevice physicalDevice,
    const VkDeviceCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkDevice* pDevice);

typedef struct VkDeviceCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkDeviceCreateFlags flags; = 0
    uint32_t queueCreateInfoCount;
    const VkDeviceQueueCreateInfo* pQueueCreateInfos;
    uint32_t enabledLayerCount;
    const char* const* ppEnabledLayerNames;
    uint32_t enabledExtensionCount;
    const char* const* ppEnabledExtensionNames;
    const VkPhysicalDeviceFeatures* pEnabledFeatures; P.14
} VkDeviceCreateInfo;
```

pNext must be NULL or point to one of:

VkDeviceGroupDeviceCreateInfo P.12
VkPhysicalDevice16BitStorageFeatures P.14
VkPhysicalDeviceFeatures2 P.14
VkPhysicalDeviceMultiviewFeatures P.14
VkPhysicalDeviceProtectedMemoryFeatures P.15
VkPhysicalDeviceSamplerYcbcrConversionFeatures P.15
VkPhysicalDeviceVariablePointerFeatures P.15

```
typedef struct VkDeviceGroupDeviceCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    uint32_t physicalDeviceCount;
    const VkPhysicalDevice* pPhysicalDevices;
} VkDeviceGroupDeviceCreateInfo;
```

Device Destruction [4.2.4]

```
void vkDestroyDevice(
    VkDevice device,
    const VkAllocationCallbacks* pAllocator); P.12
```

Command Buffers [5]

Also see Command Buffer Lifecycle diagram. P.5

Command Pools [5.2]

```
VkResult vkCreateCommandPool(
    VkDevice device,
    const VkCommandPoolCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkCommandPool* pCommandPool);

typedef struct VkCommandPoolCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkCommandPoolCreateFlags flags;
    uint32_t queueFamilyIndex;
} VkCommandPoolCreateInfo;
```

flags: VK_COMMAND_POOL_CREATE_*X*_BIT where *X* is PROTECTED, RESET_COMMAND_BUFFER, TRANSIENT

```
void vkTrimCommandPool(VkDevice device,
    VkCommandPool commandPool,
    VkCommandPoolTrimFlags flags); = 0

VkResult vkResetCommandPool(
    VkDevice device, VkCommandPool commandPool,
    VkCommandPoolResetFlags flags);
```

flags: VK_COMMAND_POOL_RESET_RELEASE_RESOURCES_BIT

```
void vkDestroyCommandPool(
    VkDevice device, VkCommandPool commandPool,
    const VkAllocationCallbacks* pAllocator); P.12
```

Continued on next page >

Command Buffers (continued)

Command Buffer Lifetime [5.3]

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

typedef struct VkCommandBufferAllocateInfo{
    VkStructureType sType; P.15
    const void* pNext;
    VkCommandPool commandPool;
    VkCommandBufferLevel level;
    uint32_t commandBufferCount;
} VkCommandBufferAllocateInfo;
  level:
    VK_COMMAND_BUFFER_LEVEL_{PRIMARY, SECONDARY}

VkResult vkResetCommandBuffer(
    VkCommandBuffer commandBuffer,
    VkCommandBufferResetFlags flags);
  flags:
    VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT

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

Command Buffer Recording [5.4]

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

typedef struct VkCommandBufferBeginInfo{
    VkStructureType sType; P.15
    const void* pNext;
    VkCommandBufferUsageFlags flags;
    const VkCommandBufferInheritanceInfo* pInheritanceInfo;
} VkCommandBufferBeginInfo;
  flags: VK_COMMAND_BUFFER_USAGE_X _BIT where X is
    ONE_TIME_SUBMIT, RENDER_PASS_CONTINUE,
    SIMULTANEOUS_USE
  pNext must be NULL or point to:
    VkDeviceGroupCommandBufferBeginInfo P.12

typedef struct VkCommandBufferInheritanceInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkRenderPass renderPass;
    uint32_t subpass;
    VkFramebuffer framebuffer;
    VkBool32 occlusionQueryEnable;
    VkQueryControlFlags queryFlags;
    VkQueryPipelineStatisticFlags pipelineStatistics; P.15
} VkCommandBufferInheritanceInfo;
    queryFlags: VK_QUERY_CONTROL_PRECISE_BIT

VkResult vkEndCommandBuffer(
    VkCommandBuffer commandBuffer);
```

Command Buffer Submission [5.5]

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

typedef struct VkSubmitInfo{
    VkStructureType sType; P.15
    const void* pNext;
    uint32_t waitSemaphoreCount;
    const VkSemaphore* pWaitSemaphores;
    const VkPipelineStageFlags* pWaitDstStageMask; P.15
    uint32_t commandBufferCount;
    const VkCommandBuffer* pCommandBuffers;
    uint32_t signalSemaphoreCount;
    const VkSemaphore* pSignalSemaphores;
} VkSubmitInfo;
    pNext must be NULL or point to one of:
      VkDeviceGroupSubmitInfo P.12
      VkProtectedSubmitInfo P.15
```

Secondary Command Buffer Execution [5.7]

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

Command Buffer Device Mask [5.8]

```
void vkCmdSetDeviceMask(
    VkCommandBuffer commandBuffer,
    uint32_t deviceMask);
```

Version Number Macros [Appendix D]

The vulkan.h header defines C preprocessor macros that are described below. Version numbers are packed into integers.

```
#define VK_VERSION_MAJOR(version) \
    ((uint32_t)(version) >> 22)

#define VK_VERSION_MINOR(version) \
    (((uint32_t)(version) >> 12) & 0x3ff)

#define VK_VERSION_PATCH(version) \
    ((uint32_t)(version) & 0xfff)

#define VK_API_VERSION_1_0 VK_MAKE_VERSION(1, 0, 0)

#define VK_API_VERSION_1_1 VK_MAKE_VERSION(1, 1, 0)

#define VK_MAKE_VERSION(major, minor, patch) \
    (((major) << 22) | ((minor) << 12) | (patch))
      major: the major version number
      minor: the minor version number
      patch: the patch version number

#define VK_HEADER_VERSION patch-version
      patch-version: synchronized with the patch version of the
        released specification
```

VK_NULL_HANDLE

This is a canonical invalid non-dispatchable handle. It is returned by certain object creation functions if creation fails, and can be passed as a parameter where permitted by VU rules.

Notes

Synchronization and Cache Control [6]

Fences [6.3]

Fence status is always either *signaled* or *unsignaled*.

```
VkResult vkCreateFence(
    VkDevice device, const VkFenceCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkFence* pFence);

typedef struct VkFenceCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkFenceCreateFlags flags;
} VkFenceCreateInfo;
  flags: VK_FENCE_CREATE_SIGNALED_BIT
  pNext must be NULL or point to:
    VkExportFenceCreateInfo P.12

void vkDestroyFence(
    VkDevice device, VkFence fence,
    const VkAllocationCallbacks* pAllocator); P.12

VkResult vkGetFenceStatus(
    VkDevice device, VkFence fence);

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

VkResult vkWaitForFences(VkDevice device,
    uint32_t fenceCount, const VkFence* pFences,
    VkBool32 waitAll, uint64_t timeout);
```

Semaphores [6.4]

Semaphore status is always either *signaled* or *unsignaled*.

```
VkResult vkCreateSemaphore(
    VkDevice device,
    const VkSemaphoreCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkSemaphore* pSemaphore);

typedef struct VkSemaphoreCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkSemaphoreCreateFlags flags; = 0
} VkSemaphoreCreateInfo;
  pNext must be NULL or point to:
    VkExportSemaphoreCreateInfo P.12

void vkDestroySemaphore(
    VkDevice device, VkSemaphore semaphore,
    const VkAllocationCallbacks* pAllocator); P.12
```

Events [6.5]

Events represent a fine-grained synchronization primitive that can be used to gauge progress through a sequence of commands executed on a queue.

```
VkResult vkCreateEvent(
    VkDevice device, const VkEventCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkEvent* pEvent);

typedef struct VkEventCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkEventCreateFlags flags; = 0
} VkEventCreateInfo;

void vkDestroyEvent(VkDevice device, VkEvent event,
    const VkAllocationCallbacks* pAllocator); P.12

VkResult vkGetEventStatus(
    VkDevice device, VkEvent event);

VkResult vk[Set, Reset]Event(
    VkDevice device, VkEvent event);

VkResult vkCmd[Set, Reset]Event(
    VkCommandBuffer commandBuffer, VkEvent event,
    VkPipelineStageFlags stageMask); P.15

void vkCmdWaitEvents(
    VkCommandBuffer commandBuffer,
    uint32_t eventCount,
    const VkEvent* pEvents,
    VkPipelineStageFlags srcStageMask, P.15
    VkPipelineStageFlags dstStageMask, P.15
    uint32_t memoryBarrierCount,
    const VkMemoryBarrier* pMemoryBarriers, P.13
    uint32_t bufferMemoryBarrierCount,
    const VkBufferMemoryBarrier*
      pBufferMemoryBarriers, P.12
    uint32_t imageMemoryBarrierCount,
    const VkImageMemoryBarrier*
      pImageMemoryBarriers); P.13
```

Pipeline Barriers [6.6]

```
void vkCmdPipelineBarrier(
    VkCommandBuffer commandBuffer,
    VkPipelineStageFlags srcStageMask, P.15
    VkPipelineStageFlags dstStageMask, P.15
    VkDependencyFlags dependencyFlags, P.1512
    uint32_t memoryBarrierCount,
    const VkMemoryBarrier* pMemoryBarriers, P.13
    uint32_t bufferMemoryBarrierCount,
    const VkBufferMemoryBarrier*
      pBufferMemoryBarriers, P.12
    uint32_t imageMemoryBarrierCount,
    const VkImageMemoryBarrier*
      pImageMemoryBarriers); P.13
```

Wait Idle Operations [6.8]

```
VkResult vkQueueWaitIdle(VkQueue queue);

VkResult vkDeviceWaitIdle(VkDevice device);
```

Render Pass [7]

A render pass represents a collection of attachments, subpasses, and dependencies between the subpasses, and describes how the attachments are used over the course of the subpasses.

Render Pass Creation [7.1]

```
VkResult vkCreateRenderPass(VkDevice device,
    const VkRenderPassCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkRenderPass* pRenderPass);
```

```
typedef struct VkRenderPassCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkRenderPassCreateFlags flags; = 0
    uint32_t attachmentCount;
    const VkAttachmentDescription* pAttachments;
    uint32_t subpassCount;
    const VkSubpassDescription* pSubpasses;
    uint32_t dependencyCount;
    const VkSubpassDependency* pDependencies;
} VkRenderPassCreateInfo;
```

pNext must be NULL or point to one of:
VkRenderPassInputAttachmentAspectCreateInfo P.1315
VkRenderPassMultiviewCreateInfo P.1315

```
typedef struct VkAttachmentDescription {
    VkAttachmentDescriptionFlags flags;
    VkFormat format; P.13
    VkSampleCountFlagBits samples; P.15
    VkAttachmentLoadOp loadOp;
    VkAttachmentStoreOp storeOp;
    VkAttachmentLoadOp stencilLoadOp;
    VkAttachmentStoreOp stencilStoreOp;
    VkImageLayout initialLayout; P.13
    VkImageLayout finalLayout; P.13
} VkAttachmentDescription;
```

flags: VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT

loadOp, stencilLoadOp: VK_ATTACHMENT_LOAD_OP_*X* where *X* is LOAD, CLEAR, DONT_CARE

storeOp, stencilStoreOp: VK_ATTACHMENT_STORE_OP_*X* where *X* is STORE, DONT_CARE

```
typedef struct VkSubpassDescription {
    VkSubpassDescriptionFlags flags;
    VkPipelineBindPoint pipelineBindPoint; P.1315
    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;
```

```
typedef struct VkAttachmentReference {
    uint32_t attachment;
    VkImageLayout layout; P.13
} VkAttachmentReference;
```

```
typedef struct VkSubpassDependency {
    uint32_t srcSubpass;
    uint32_t dstSubpass;
    VkPipelineStageFlags srcStageMask; P.15
    VkPipelineStageFlags dstStageMask; P.12
    VkAccessFlags srcAccessMask; P.12
    VkAccessFlags dstAccessMask; P.12
    VkDependencyFlags dependencyFlags; P.1512
} VkSubpassDependency;
```

```
void vkDestroyRenderPass(VkDevice device,
    VkRenderPass renderPass,
    const VkAllocationCallbacks* pAllocator); P.12
```

Framebuffers [7.3]

```
VkResult vkCreateFramebuffer(VkDevice device,
    const VkFramebufferCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkFramebuffer* pFramebuffer);
```

```
typedef struct VkFramebufferCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkFramebufferCreateFlags flags; = 0
    VkRenderPass renderPass;
    uint32_t attachmentCount;
    const VkImageView* pAttachments;
    uint32_t width;
    uint32_t height;
    uint32_t layers;
} VkFramebufferCreateInfo;
```

```
void vkDestroyFramebuffer(
    VkDevice device, VkFramebuffer framebuffer,
    const VkAllocationCallbacks* pAllocator); P.12
```

Render Pass Commands [7.4]

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

contents: VK_SUBPASS_CONTENTS_*X* where *X* is INLINE, SECONDARY_COMMAND_BUFFERS

```
typedef struct VkRenderPassBeginInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkRenderPass renderPass;
    VkFramebuffer framebuffer;
    VkRect2D renderArea; P.15
    uint32_t clearValueCount;
    const VkClearValue* pClearValues; P.12
} VkRenderPassBeginInfo;
```

pNext must be NULL or point to:
VkDeviceGroupRenderPassBeginInfo P.12

```
void vkGetRenderAreaGranularity(
    VkDevice device, VkRenderPass renderPass,
    VkExtent2D* pGranularity); P.12
```

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

contents: VK_SUBPASS_CONTENTS_*X* where *X* is INLINE, SECONDARY_COMMAND_BUFFERS

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

Shaders [8]

Shader Modules [8.1]

```
VkResult vkCreateShaderModule(
    VkDevice device,
    const VkShaderModuleCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkShaderModule* pShaderModule);
```

```
typedef struct VkShaderModuleCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkShaderModuleCreateFlags flags; = 0
    size_t codeSize;
    const uint32_t* pCode;
} VkShaderModuleCreateInfo;
```

```
void vkDestroyShaderModule(
    VkDevice device,
    VkShaderModule shaderModule,
    const VkAllocationCallbacks* pAllocator); P.12
```

Built-in Variables [14.6]

The built-in variables listed below are accessed in shaders by declaring the variable using a **BuiltIn** decoration.

Decoration	Type
BaseInstance	Scalar 32-bit integer
BaseVertex	Scalar 32-bit integer
ClipDistance	Array of 32-bit floats
CullDistance	Array of 32-bit floats
DeviceIndex	Scalar 32-bit integer
DrawIndex	Scalar 32-bit integer
FragCoord	4-component vector of 32-bit floats
FragDepth	Scalar 32-bit float
FrontFacing	Scalar 32-bit integer
GlobalInvocationID	3-component vector of 32-bit ints
HelperInvocation	Scalar 32-bit integer
InvocationID	Scalar 32-bit integer
InstanceIndex	Scalar 32-bit integer
Layer	Scalar 32-bit integer
LocalInvocationID	3-component vector of 32-bit ints
NumSubgroups	Scalar 32-bit integer
NumWorkGroups	3-component vector of 32-bit ints
PatchVertices	Scalar 32-bit integer
PointCoord	2-component vector of 32-bit floats
PointSize	Scalar 32-bit float value
Position	4-component vector of 32-bit floats
PrimitiveID	Scalar 32-bit integer
SampleID	Scalar 32-bit integer
SampleMask	Array of 32-bit integers
SamplePosition	2-component vector of float values
SubgroupId	Scalar 32-bit integer
Subgroup{Eq,Ge,Gt,Le,Lt}Mask	4-component vector of 32-bit ints
SubgroupLocalInvocationId	Scalar 32-bit integer
SubgroupSize	Scalar 32-bit integer
TessCoord	3-component vector of 32-bit floats
TessLevelOuter	Array of size 2 of 32-bit floats
TessLevelInner	Array of size 4 of 32-bit floats
VertexIndex	32-bit integer
ViewIndex	Scalar 32-bit integer
ViewportIndex	32-bit integer
WorkgroupSize	3-component vector of 32-bit ints
WorkgroupID	3-component vector of 32-bit ints

Pipelines [9]

Compute Pipelines [9.1]

Compute pipelines consist of a single static compute shader stage and the pipeline layout.

```
VkResult vkCreateComputePipelines(
    VkDevice device, VkPipelineCache pipelineCache,
    uint32_t createInfoCount,
    const VkComputePipelineCreateInfo* pCreateInfos,
    const VkAllocationCallbacks* pAllocator, P.12
    VkPipeline* pPipelines);
```

```
typedef struct VkComputePipelineCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineCreateFlags flags; P.15
    VkPipelineShaderStageCreateInfo stage; P.15
    VkPipelineLayout layout;
    VkPipeline basePipelineHandle;
    int32_t basePipelineIndex;
} VkComputePipelineCreateInfo;
```

Graphics Pipelines [9.2]

```
VkResult vkCreateGraphicsPipelines(
    VkDevice device, VkPipelineCache pipelineCache,
    uint32_t createInfoCount,
    const VkGraphicsPipelineCreateInfo* pCreateInfos,
    const VkAllocationCallbacks* pAllocator, P.12
    VkPipeline* pPipelines);
```

In VkGraphicsPipelineCreateInfo below, replace *X* with VkPipeline and replace *Y* with StateCreateInfo. For example, *X*VertexInput*Y* would be VxPipelineVertexInputStateCreateInfo.

```
typedef struct VkGraphicsPipelineCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineCreateFlags flags; P.15
    uint32_t stageCount;
    const VkPipelineShaderStageCreateInfo* pStages; P.15
    const XVertexInputY* pVertexInputState;
    const XInputAssemblyY* pInputAssemblyState;
    const XTessellationY* pTessellationState;
    const XViewportY* pViewportState;
    const XRasterizationY* pRasterizationState;
    const XMultisampleY* pMultisampleState;
    const XDepthStencilY* pDepthStencilState;
    const XColorBlendY* pColorBlendState;
    const XDynamicY* pDynamicState;
    VkPipelineLayout layout;
    VkRenderPass renderPass;
    uint32_t subpass;
    VkPipeline basePipelineHandle;
    int32_t basePipelineIndex;
} VkGraphicsPipelineCreateInfo;
```

```
typedef struct VkPipelineVertexInputStateCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineVertexInputStateCreateFlags flags; = 0
    uint32_t vertexBindingDescriptionCount;
    const VkVertexInputBindingDescription*
        pVertexBindingDescriptions;
    uint32_t vertexAttributeDescriptionCount;
    const VkVertexInputAttributeDescription*
        pVertexAttributeDescriptions;
} VkPipelineVertexInputStateCreateInfo;
```

```
typedef struct VkVertexInputBindingDescription {
    uint32_t binding;
    uint32_t stride;
    VkVertexInputRate inputRate;
} VkVertexInputBindingDescription;
```

inputRate:
VK_VERTEX_INPUT_RATE_{VERTEX, INSTANCE}

Continued on next page >

Pipelines (continued)

```
typedef struct VkVertexInputAttributeDescription {
    uint32_t location;
    uint32_t binding;
    VkFormat format; P.13
    uint32_t offset;
} VkVertexInputAttributeDescription;
```

```
typedef struct VkPipelineInputAssemblyStateCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineInputAssemblyStateCreateFlags flags; = 0
    VkPrimitiveTopology topology;
    VkBool32 primitiveRestartEnable;
} VkPipelineInputAssemblyStateCreateInfo;
```

topology: VK_PRIMITIVE_TOPOLOGY_*X* where *X* is POINT_LIST, LINE_LIST, LINE_STRIP, TRIANGLE_LIST, TRIANGLE_STRIP, TRIANGLE_FAN, LINE_{LIST, STRIP}_WITH_ADJACENCY, TRIANGLE_{LIST, STRIP}_WITH_ADJACENCY, PATCH_LIST

```
typedef struct VkPipelineTessellationStateCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineTessellationStateCreateFlags flags; = 0
    uint32_t patchControlPoints;
} VkPipelineTessellationStateCreateInfo;
```

pNext must be NULL or point to:
VkPipelineTessellationDomainOriginStateCreateInfo P.15

```
typedef struct VkPipelineViewportStateCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineViewportStateCreateFlags flags; = 0
    uint32_t viewportCount;
    const VkViewport* pViewports; P.15
    uint32_t scissorCount;
    const VkRect2D* pScissors; P.15
} VkPipelineViewportStateCreateInfo;
```

```
typedef struct VkPipelineRasterizationStateCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineRasterizationStateCreateFlags flags; = 0
    VkBool32 depthClampEnable;
    VkBool32 rasterizerDiscardEnable;
    VkPolygonMode polygonMode;
    VkCullModeFlags cullMode;
    VkFrontFace frontFace;
    VkBool32 depthBiasEnable;
    float depthBiasConstantFactor;
    float depthBiasClamp;
    float depthBiasSlopeFactor;
    float lineWidth;
} VkPipelineRasterizationStateCreateInfo;
```

polygonMode: VK_POLYGON_MODE_{FILL, LINE, POINT}

cullMode: VK_CULL_MODE_*X* where *X* is NONE, FRONT_BIT, BACK_BIT, FRONT_AND_BACK

frontFace: VK_FRONT_FACE_[COUNTER_]CLOCKWISE

```
typedef struct VkPipelineMultisampleStateCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineMultisampleStateCreateFlags flags; = 0
    VkSampleCountFlagBits rasterizationSamples; P.15
    VkBool32 sampleShadingEnable;
    float minSampleShading;
    const VkSampleMask* pSampleMask;
    VkBool32 alphaToCoverageEnable;
    VkBool32 alphaToOneEnable;
} VkPipelineMultisampleStateCreateInfo;
```

```
typedef struct VkPipelineDepthStencilStateCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineDepthStencilStateCreateFlags flags; = 0
    VkBool32 depthTestEnable;
    VkBool32 depthWriteEnable;
    VkCompareOp depthCompareOp; P.12
    VkBool32 depthBoundsTestEnable;
    VkBool32 stencilTestEnable;
    VkStencilOpState front;
    VkStencilOpState back;
    float minDepthBounds;
    float maxDepthBounds;
} VkPipelineDepthStencilStateCreateInfo;
```

```
typedef struct VkStencilOpState {
    VkStencilOp failOp;
    VkStencilOp passOp;
    VkStencilOp depthFailOp;
    VkCompareOp compareOp; P.12
    uint32_t compareMask;
    uint32_t writeMask;
    uint32_t reference;
} VkStencilOpState;
```

enum VkStencilOp: VK_STENCIL_OP_*X* where *X* is KEEP, ZERO, REPLACE, INCREMENT_AND_{CLAMP, WRAP}, INVERT, DECREMENT_AND_{CLAMP, WRAP}

```
typedef struct VkPipelineColorBlendStateCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineColorBlendStateCreateFlags flags; = 0
    VkBool32 logicOpEnable;
    VkLogicOp logicOp;
    uint32_t attachmentCount;
    const VkPipelineColorBlendAttachmentState*
        pAttachments;
    float blendConstants[4];
} VkPipelineColorBlendStateCreateInfo;
```

logicOp: VK_LOGIC_OP_*X* where *X* is CLEAR, AND, AND_REVERSE, COPY, AND_INVERTED, NO_OP, XOR, OR, NOR, EQUIVALENT, INVERT, OR_REVERSE, COPY_INVERTED, OR_INVERTED, NAND, SET

blendOp: VK_BLEND_OP_*X* where *X* is ADD, SUBTRACT, REVERSE_SUBTRACT, MIN, MAX

colorWriteMask: VK_COLOR_COMPONENT_*X* where *X* is R_BIT, G_BIT, B_BIT, A_BIT

```
typedef struct VkPipelineColorBlendAttachmentState {
    VkBool32 blendEnable;
    VkBlendFactor srcColorBlendFactor;
    VkBlendFactor dstColorBlendFactor;
    VkBlendOp colorBlendOp; P.12
    VkBlendFactor srcAlphaBlendFactor;
    VkBlendFactor dstAlphaBlendFactor;
    VkBlendOp alphaBlendOp; P.12
    VkColorComponentFlags colorWriteMask;
} VkPipelineColorBlendAttachmentState;
```

enum VkBlendFactor: VK_BLEND_FACTOR_*X* where *X* is ZERO, ONE, SRC_ALPHA_SATURATE, [ONE_MINUS_]SRC_COLOR, [ONE_MINUS_]DST_COLOR, [ONE_MINUS_]SRC_ALPHA, [ONE_MINUS_]DST_ALPHA, [ONE_MINUS_]CONSTANT_COLOR, [ONE_MINUS_]CONSTANT_ALPHA, [ONE_MINUS_]SRC1_COLOR, [ONE_MINUS_]SRC1_ALPHA

colorWriteMask:
VK_COLOR_COMPONENT_*X*_BIT where *X* is R, G, B, A

```
typedef struct VkPipelineDynamicStateCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineDynamicStateCreateFlags flags; = 0
    uint32_t dynamicStateCount;
    const VkDynamicState* pDynamicStates;
} VkPipelineDynamicStateCreateInfo;
```

pDynamicStates: Array of VK_DYNAMIC_STATE_*X* where *X* is VIEWPORT, SCISSOR, LINE_WIDTH, DEPTH_BIAS, BLEND_CONSTANTS, DEPTH_BOUNDS, STENCIL_REFERENCE, STENCIL_COMPARE_MASK, STENCIL_WRITE_MASK

Pipeline Destruction [9.3]

```
void vkDestroyPipeline(
    VkDevice device, VkPipeline pipeline,
    const VkAllocationCallbacks* pAllocator); P.12
```

Pipeline Cache [9.6]

Pipeline cache objects allow the result of pipeline construction to be reused between pipelines and between runs of an application.

```
VkResult vkCreatePipelineCache(VkDevice device,
    const VkPipelineCacheCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkPipelineCache* pPipelineCache);
```

```
typedef struct VkPipelineCacheCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineCacheCreateFlags flags; = 0
    size_t initialDataSize;
    const void* pInitialData;
} VkPipelineCacheCreateInfo;
```

```
VkResult vkMergePipelineCaches(VkDevice device,
    VkPipelineCache dstCache, uint32_t srcCacheCount,
    const VkPipelineCache* pSrcCaches);
```

```
VkResult vkGetPipelineCacheData(VkDevice device,
    VkPipelineCache pipelineCache,
    size_t* pDataSize, void* pData);
```

```
void vkDestroyPipelineCache(VkDevice device,
    VkPipelineCache pipelineCache,
    const VkAllocationCallbacks* pAllocator); P.12
```

Pipeline Binding [9.8]

```
void vkCmdBindPipeline(
    VkCommandBuffer commandBuffer,
    VkPipelineBindPoint pipelineBindPoint, P.13
    VkPipeline pipeline);
```

Memory Allocation [10]

Device Memory [10.2]

Device memory is memory that is visible to the device.

```
void vkGetPhysicalDeviceMemoryProperties(
    VkPhysicalDevice physicalDevice,
    VkPhysicalDeviceMemoryProperties*
        pMemoryProperties); P.14
```

```
void vkGetPhysicalDeviceMemoryProperties2(
    VkPhysicalDevice physicalDevice,
    VkPhysicalDeviceMemoryProperties2*
        pMemoryProperties);
```

```
typedef struct VkPhysicalDeviceMemoryProperties2 {
    VkStructureType sType; P.15
    void* pNext;
    VkPhysicalDeviceMemoryProperties
        memoryProperties; P.14
} VkPhysicalDeviceMemoryProperties2;
```

```
VkResult vkAllocateMemory(
    VkDevice device,
    const VkMemoryAllocateInfo* pAllocateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkDeviceMemory* pMemory);
```

```
typedef struct VkMemoryAllocateInfo {
    VkStructureType sType; P.15 const void* pNext;
    VkDeviceSize* allocationSize; uint32_t memoryTypeIndex;
} VkMemoryAllocateInfo;
```

pNext must be NULL or point to one of:
VkExportMemoryAllocateInfo P.12
VkMemoryAllocateFlagsInfo P.13
VkMemoryDedicatedAllocateInfo P.13

```
void vkFreeMemory(
    VkDevice device,
    VkDeviceMemory memory,
    const VkAllocationCallbacks* pAllocator); P.12
```

Host Access to Device Memory Objects [10.2.1]

Memory objects created with vkAllocateMemory are not directly host accessible. Memory objects created with memory property VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT are considered mappable. Memory objects must be mappable in order to be successfully mapped on the host.

```
VkResult vkMapMemory(
    VkDevice device,
    VkDeviceMemory memory,
    VkDeviceSize offset,
    VkDeviceSize size,
    VkMemoryMapFlags flags, = 0
    void** ppData);
```

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

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

```
typedef struct VkMappedMemoryRange {
    VkStructureType sType; P.15
    const void* pNext;
    VkDeviceMemory memory;
    VkDeviceSize offset; VkDeviceSize size;
} VkMappedMemoryRange;
```

```
void vkUnmapMemory(
    VkDevice device,
    VkDeviceMemory memory);
```

Lazily Allocated Memory [10.2.2]

If the memory object is allocated from a heap with the VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT bit set, that object's backing memory may be provided by the implementation lazily.

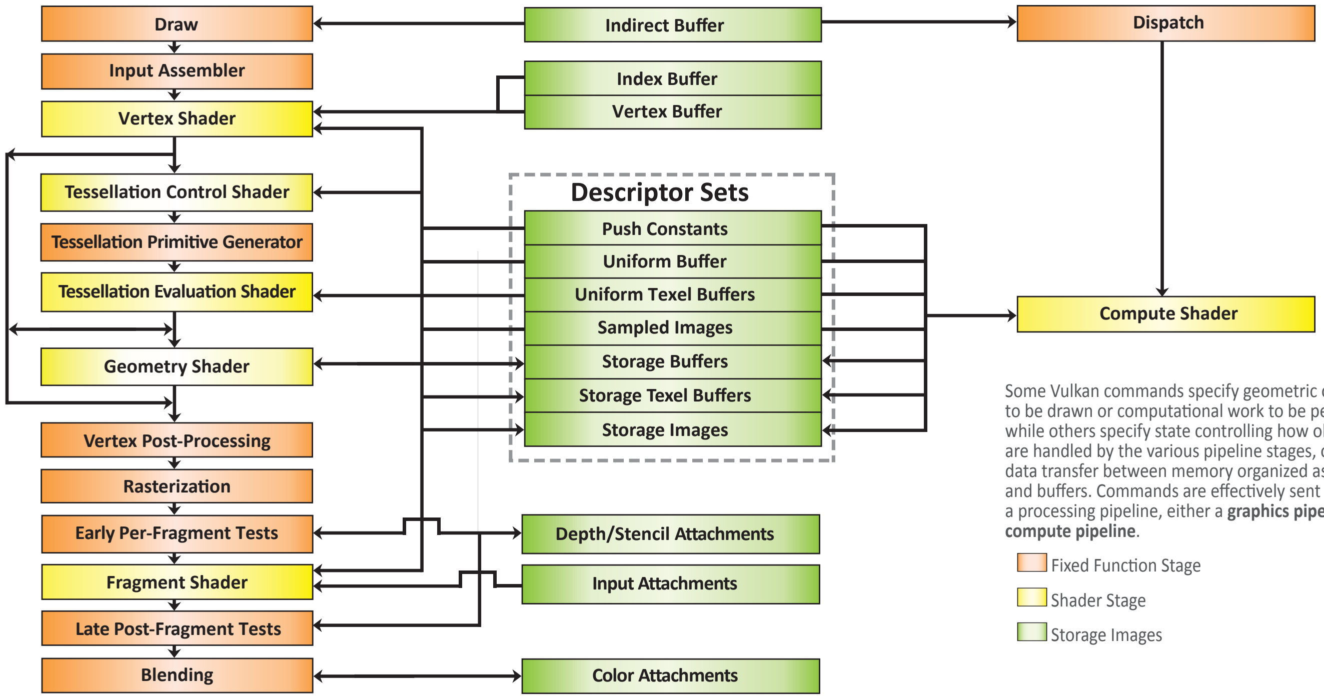
```
void vkGetDeviceMemoryCommitment(
    VkDevice device,
    VkDeviceMemory memory,
    VkDeviceSize* pCommittedMemoryInBytes);
```

Peer Memory Features [10.2.4]

```
void vkGetDeviceGroupPeerMemoryFeatures(
    VkDevice device, uint32_t heapIndex,
    uint32_t localDeviceIndex, uint32_t remoteDeviceIndex,
    VkPeerMemoryFeatureFlags* pPeerMemoryFeatures);
```

pPeerMemoryFeatures: VK_PEER_MEMORY_FEATURE_*X* where *X* is COPY_SRC_BIT, COPY_DST_BIT, GENERIC_SRC_BIT, GENERIC_DST_BIT

Vulkan Pipeline Diagram [9]



Draw
Input Assembler
Vertex Shader
Tessellation Control Shader
Tessellation Primitive Generator
Tessellation Evaluation Shader
Geometry Shader
Vertex Post-Processing
Rasterization
Early Per-Fragment Tests
Fragment Shader
Late Post-Fragment Tests
Blending

Indirect Buffer
Index Buffer
Vertex Buffer

Descriptor Sets
Push Constants
Uniform Buffer
Uniform Texel Buffers
Sampled Images
Storage Buffers
Storage Texel Buffers
Storage Images

Depth/Stencil Attachments
Input Attachments
Color Attachments

Dispatch
Compute Shader

Some Vulkan commands specify geometric objects to be drawn or computational work to be performed, while others specify state controlling how objects are handled by the various pipeline stages, or control data transfer between memory organized as images and buffers. Commands are effectively sent through a processing pipeline, either a **graphics pipeline** or a **compute pipeline**.

🟧 Fixed Function Stage
🟨 Shader Stage
🟩 Storage Images

Command Buffer Lifecycle [5.1]

A command buffer is always in one of the five states shown below:

Initial state

The state when a command buffer is first allocated. The command buffer may be reset back to this state from any of the executable, recording, or invalid states. Command buffers in the initial state can only be moved to recording, or freed.

Recording state

vkBeginCommandBuffer changes the state from initial to recording. Once in the recording state, **vkCmd*** commands can be used to record to the command buffer.

Executable state

vkEndCommandBuffer moves a command buffer state from recording to executable.

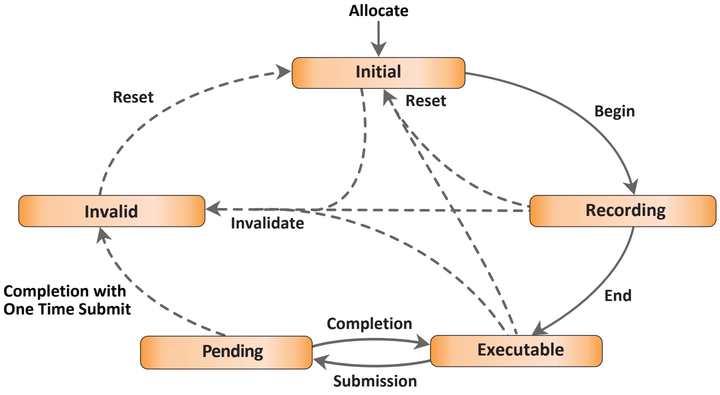
Executable command buffers can be submitted, reset, or recorded to another command buffer.

Pending state

Queue submission changes the state from executable to pending, in which applications must not attempt to modify the command buffer in any way. The state reverts back to executable when current executions complete, or to invalid.

Invalid state

Some operations will transition the command buffer into the invalid state, in which it can only be reset or freed.



Resource Creation [11]

Buffers [11.1]

Buffers represent linear arrays of data which are used for various purposes by binding them to a graphics or compute pipeline via descriptor sets or via certain commands, or by directly specifying them as parameters to certain commands.

```
VkResult vkCreateBuffer(
    VkDevice device,
    const VkBufferCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkBuffer* pBuffer);
```

```
typedef struct VkBufferCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkBufferCreateFlags flags;
    VkDeviceSize size;
    VkBufferUsageFlags usage;   P.12
    VkSharingMode sharingMode; P.15
    uint32_t queueFamilyIndexCount;
    const uint32_t* pQueueFamilyIndices;
} VkBufferCreateInfo;
```

flags:
VK_BUFFER_CREATE_SPARSE_*X*_BIT where *X* is ALIASED, BINDING, PROTECTED, RESIDENCY

pNext must be NULL or point to:
VkExternalMemoryBufferCreateInfo P.12

```
void vkDestroyBuffer(
    VkDevice device,
    VkBuffer buffer,
    const VkAllocationCallbacks* pAllocator); P.12
```

Buffer Views [11.2]

A buffer view represents a contiguous range of a buffer and a specific format to be used to interpret the data.

```
VkResult vkCreateBufferView(
    VkDevice device,
    const VkBufferViewCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkBufferView* pView);
```

```
typedef struct VkBufferViewCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkBufferViewCreateFlags flags; = 0
    VkBuffer buffer;
    VkFormat format; P.13
    VkDeviceSize offset; VkDeviceSize range;
} VkBufferViewCreateInfo;
```

```
void vkDestroyBufferView(
    VkDevice device,
    VkBufferView bufferView,
    const VkAllocationCallbacks* pAllocator); P.12
```

Images [11.3]

Images represent multidimensional (up to 3) arrays of data which can be used for various purposes by binding them to the graphics or compute pipeline via descriptor sets, or by directly specifying them as parameters to certain commands.

```
VkResult vkCreateImage(
    VkDevice device,
    const VkImageCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkImage* pImage);
```

```
typedef struct VkImageCreateInfo {
    VkStructureType sType;  P.15
    const void* pNext;
    VkImageCreateFlags flags; P.13
    VkImageType imageType; P.13
    VkFormat format; P.13
    VkExtent3D extent; PP.12
    uint32_t mipLevels; uint32_t arrayLayers;
    VkSampleCountFlagBits samples; P.15
    VkImageTiling tiling; P.13
    VkImageUsageFlags usage; P.13
    VkSharingMode sharingMode; P.15
    uint32_t queueFamilyIndexCount;
    const uint32_t* pQueueFamilyIndices;
    VkImageLayout initialLayout;  P.13
} VkImageCreateInfo;
```

pNext must be NULL or point to:
VkExternalMemoryImageCreateInfo P.13

```
typedef struct VkImageSwapchainCreateInfoKHR {
    VkStructureType sType;  P.15
    const void* pNext;
    VkSwapchainKHR swapchain;
} VkImageSwapchainCreateInfoKHR;
```

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

```
typedef struct VkImageSubresource {
    VkImageAspectFlags aspectMask;  P.13
    uint32_t mipLevel;
    uint32_t arrayLayer;
} VkImageSubresource;
```

Continued on next page >

Resource Creation (continued)

```
typedef struct VkSubresourceLayout {
    VkDeviceSize offset;
    VkDeviceSize size;
    VkDeviceSize rowPitch;
    VkDeviceSize arrayPitch;
    VkDeviceSize depthPitch;
} VkSubresourceLayout;
```

```
void vkDestroyImage(
    VkDevice device, VkImage image,
    const VkAllocationCallbacks* pAllocator); P.12
```

Image Views [11.5]

Image objects are not directly accessed by pipeline shaders for reading or writing image data. Instead, image views representing contiguous ranges of the image subresources and containing additional metadata are used for that purpose.

```
VkResult vkCreateImageView(
    VkDevice device,
    const VkImageViewCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkImageView* pView);
```

```
typedef struct VkImageViewCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkImageViewCreateFlags flags; = 0
    VkImage image;
    VkImageViewType viewType;
    VkFormat format; P.13
    VkComponentMapping components;
    VkImageSubresourceRange subresourceRange; P.13
} VkImageViewCreateInfo;
```

viewType: VK_IMAGE_VIEW_TYPE_*X* where *X* is 1D, 2D, 3D, CUBE, 1D_ARRAY, 2D_ARRAY, CUBE_ARRAY

pNext must be NULL or point to one of:
VkImageViewUsageCreateInfo P.13
VkSamplerYcbcrConversionInfo P.15

```
typedef struct VkComponentMapping {
    VkComponentSwizzle r;
    VkComponentSwizzle g;
    VkComponentSwizzle b;
    VkComponentSwizzle a;
} VkComponentMapping;
```

enum VkComponentSwizzle: VK_COMPONENT_SWIZZLE_*X* where *X* is IDENTITY, ZERO, ONE, R, G, B, A

```
void vkDestroyImageView(VkDevice device,
    VkImageView imageView,
    const VkAllocationCallbacks* pAllocator); P.12
```

Resource Memory Association [11.6]

Resources are initially created as virtual allocations with no backing memory. Device memory is allocated separately and then associated with the resource.

```
void vkGetBufferMemoryRequirements(
    VkDevice device,
    VkBuffer buffer,
    VkMemoryRequirements* pMemoryRequirements); P.13
```

```
void vkGetBufferMemoryRequirements2(VkDevice device,
    const VkBufferMemoryRequirementsInfo2* pInfo,
    VkMemoryRequirements2* pMemoryRequirements); P.13
```

```
typedef struct VkBufferMemoryRequirementsInfo2 {
    VkStructureType sType; P.15
    const void* pNext;
    VkBuffer buffer;
} VkBufferMemoryRequirementsInfo2;
```

```
void vkGetImageMemoryRequirements(
    VkDevice device, VkImage image,
    VkMemoryRequirements* pMemoryRequirements); P.13
```

```
void vkGetImageMemoryRequirements2(VkDevice device,
    const VkImageMemoryRequirementsInfo2* pInfo,
    VkMemoryRequirements2* pMemoryRequirements); P.13
```

```
typedef struct VkImageMemoryRequirementsInfo2 {
    VkStructureType sType; P.15
    const void* pNext;
    VkImage image;
} VkImageMemoryRequirementsInfo2;
```

pNext must be NULL or point to:
VkImagePlaneMemoryRequirementsInfo P.13

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

```
VkResult vkBindBufferMemory2(VkDevice device,
    uint32_t bindInfoCount,
    const VkBindBufferMemoryInfo* pBindInfos);
```

```
typedef struct VkBindBufferMemoryInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkBuffer buffer;
    VkDeviceMemory memory;
    VkDeviceSize memoryOffset;
} VkBindBufferMemoryInfo;
```

pNext must be NULL or point to:
VkBindBufferMemoryDeviceGroupInfo P.12

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

```
VkResult vkBindImageMemory2(VkDevice device,
    uint32_t bindInfoCount,
    const VkBindImageMemoryInfo* pBindInfos);
```

```
typedef struct VkBindImageMemoryInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkImage image;
    VkDeviceMemory memory;
    VkDeviceSize memoryOffset;
} VkBindImageMemoryInfo;
```

pNext must be NULL or point to one of:
VkBindImageMemoryDeviceGroupInfo P.12
VkBindImagePlaneMemoryInfo P.12

Samplers [12]

VkSampler objects encapsulate the state of an image sampler which is used by the implementation to read image data and apply filtering and other transformations for the shader.

```
VkResult vkCreateSampler(
    VkDevice device,
    const VkSamplerCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkSampler* pSampler);
```

```
typedef struct VkSamplerCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkSamplerCreateFlags flags; = 0
    VkFilter magFilter; VkFilter minFilter;
    VkSamplerMipmapMode mipmapMode;
    VkSamplerAddressMode addressModeU;
    VkSamplerAddressMode addressModeV;
    VkSamplerAddressMode addressModeW;
    float mipLodBias; VkBool32 anisotropyEnable;
    float maxAnisotropy; VkBool32 compareEnable;
    VkCompareOp compareOp; P.12
    float minLod; float maxLod;
    VkBorderColor borderColor;
    VkBool32 unnormalizedCoordinates;
} VkSamplerCreateInfo;
```

magFilter, minFilter: VK_FILTER_NEAREST, VK_FILTER_LINEAR

mipmapMode:
VK_SAMPLER_MIPMAP_MODE_{NEAREST, LINEAR}

borderColor: VK_BORDER_COLOR_{FLOAT, INT}_*X* where *X* is TRANSPARENT_BLACK, OPAQUE_BLACK, OPAQUE_WHITE

addressMode{U, V, W}:
VK_SAMPLER_ADDRESS_MODE_*X* where *X* is REPEAT, MIRRORED_REPEAT, MIRROR_CLAMP_TO_EDGE, CLAMP_TO_EDGE, CLAMP_TO_BORDER

pNext must be NULL or point to:
VkSamplerYcbcrConversionInfo P.15

```
void vkDestroySampler(
    VkDevice device,
    VkSampler sampler,
    const VkAllocationCallbacks* pAllocator); P.12
```

Sampler Y'$C_B$$C_R$ Conversion [12.1]

```
VkResult vkCreateSamplerYcbcrConversion(
    VkDevice device,
    const VkSamplerYcbcrConversionCreateInfo*
      pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkSamplerYcbcrConversion* pYcbcrConversion);
```

```
typedef struct VkSamplerYcbcrConversionCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkFormat format; P.13
    VkSamplerYcbcrModelConversion ycbcrModel;
    VkSamplerYcbcrRange ycbcrRange;
    VkComponentMapping components;
    VkChromaLocation xChromaOffset;
    VkChromaLocation yChromaOffset;
    VkFilter chromaFilter;
    VkBool32 forceExplicitReconstruction;
} VkSamplerYcbcrConversionCreateInfo;
```

VkSamplerYcbcrModelConversion:
VK_SAMPLER_YCBCR_MODEL_CONVERSION_*X* where *X* is {RGB, YCBCR}_IDENTITY, YCBCR_{709, 601, 2020}

VkSamplerYcbcrRange:
VK_SAMPLER_YCBCR_RANGE_ITU_{FULL, NARROW}

VkChromaLocation:
VK_CHROMA_LOCATION_{COSITED_EVEN, MIDPOINT}

VkFilter:
VK_FILTER_{NEAREST, LINEAR}

```
void vkDestroySamplerYcbcrConversion(VkDevice device,
    VkSamplerYcbcrConversion ycbcrConversion,
    const VkAllocationCallbacks* pAllocator); P.12
```

Resource Descriptors [13]

An opaque data structure representing a shader resource e.g., a buffer view, image view, sampler, or combined image sampler.

Descriptor Set Layout [13.2.1]

```
VkResult vkCreateDescriptorSetLayout(
    VkDevice device,
    const VkDescriptorSetLayoutCreateInfo* pCreateInfo,P.12
    const VkAllocationCallbacks* pAllocator, P.12
    VkDescriptorSetLayout* pSetLayout);
```

```
void vkGetDescriptorSetLayoutSupport(
    VkDevice device,
    const VkDescriptorSetLayoutCreateInfo* pCreateInfo, P.1512
    VkDescriptorSetLayoutSupport* pSupport);
```

```
typedef struct VkDescriptorSetLayoutSupport {
    VkStructureType sType; P.15
    void* pNext; VkBool32 supported;
} VkDescriptorSetLayoutSupport;
```

```
void vkDestroyDescriptorSetLayout(
    VkDevice device,
    VkDescriptorSetLayout descriptorSetLayout,,
    const VkAllocationCallbacks* pAllocator); P.12
```

Pipeline Layouts [13.2.2]

```
VkResult vkCreatePipelineLayout(
    VkDevice device,
    const VkPipelineLayoutCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkPipelineLayout* pPipelineLayout);
```

```
typedef struct VkPipelineLayoutCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkPipelineLayoutCreateFlags flags; = 0
    uint32_t setLayoutCount;
    const VkDescriptorSetLayout* pSetLayouts;
    uint32_t pushConstantRangeCount;
    const VkPushConstantRange* pPushConstantRanges;
} VkPipelineLayoutCreateInfo;
```

```
typedef struct VkPushConstantRange {
    VkShaderStageFlags stageFlags; P.15
    uint32_t offset; uint32_t size;
} VkPushConstantRange;
```

```
void vkDestroyPipelineLayout(
    VkDevice device, VkPipelineLayout pipelineLayout,
    const VkAllocationCallbacks* pAllocator); P.12
```

Allocation of Descriptor Sets [13.2.3]

```
VkResult vkCreateDescriptorPool(
    VkDevice device,
    const VkDescriptorPoolCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkDescriptorPool* pDescriptorPool);
```

```
typedef struct VkDescriptorPoolCreateInfo {
    VkStructureType sType; P.15
    const void* pNext; VkDescriptorPoolCreateFlags flags;
    uint32_t maxSets; uint32_t poolSizeCount;
    const VkDescriptorPoolSize* pPoolSizes;
} VkDescriptorPoolCreateInfo;
```

Continued on next page >

Resource Descriptors (continued)

```
    flags:
       VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT

typedef struct VkDescriptorPoolSize {
    VkDescriptorType type; P.12
    uint32_t descriptorCount;
} VkDescriptorPoolSize;

void vkDestroyDescriptorPool(
    VkDevice device,
    VkDescriptorPool descriptorPool,
    const VkAllocationCallbacks* pAllocator); P.12

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

typedef struct VkDescriptorSetAllocateInfo {
    VkStructureType sType; P.15
    const void* pNext; VkDescriptorPool descriptorPool;
    uint32_t descriptorSetCount;
    const VkDescriptorSetLayout* pSetLayouts;
} VkDescriptorSetAllocateInfo;

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

VkResult vkResetDescriptorPool(
    VkDevice device,
    VkDescriptorPool descriptorPool,
    VkDescriptorPoolResetFlags flags);
```

Descriptor Set Updates [13.2.4]

```
void vkUpdateDescriptorSets(
    VkDevice device,
    uint32_t descriptorWriteCount,
    const VkWriteDescriptorSet* pDescriptorWrites,
    uint32_t descriptorCopyCount,
    const VkCopyDescriptorSet* pDescriptorCopies);

typedef struct VkWriteDescriptorSet {
    VkStructureType sType; P.15
    const void* pNext; VkDescriptorSet dstSet;
    uint32_t dstBinding;
    uint32_t dstArrayElement;
    uint32_t descriptorCount;
    VkDescriptorType descriptorType; P.12
    const VkDescriptorImageInfo* pImageInfo;
    const VkDescriptorBufferInfo* pBufferInfo;
    const VkBufferView* pTexelBufferView;
} VkWriteDescriptorSet;

typedef struct VkDescriptorImageInfo {
    VkSampler sampler;
    VkImageView imageView;
    VkImageLayout imageLayout; P.13
} VkDescriptorImageInfo;

typedef struct VkDescriptorBufferInfo {
    VkBuffer buffer;
    VkDeviceSize offset; VkDeviceSize range;
} VkDescriptorBufferInfo;

typedef struct VkCopyDescriptorSet {
    VkStructureType sType; P.15
    const void* pNext; VkDescriptorSet srcSet;
    uint32_t srcBinding;
    uint32_t srcArrayElement;
    VkDescriptorSet dstSet;
    uint32_t dstBinding;
    uint32_t dstArrayElement;
    uint32_t descriptorCount;
} VkCopyDescriptorSet;
```

Descriptor Set Updates with Templates [13.2.6]

```
VkResult vkCreateDescriptorUpdateTemplate(
    VkDevice device,
    const VkDescriptorUpdateTemplateCreateInfo*
       pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkDescriptorUpdateTemplate*
       pDescriptorUpdateTemplate);

typedef struct VkDescriptorUpdateTemplateCreateInfo {
    VkStructureType sType; P.15
    void* pNext;
    VkDescriptorUpdateTemplateCreateFlags flags; = 0
    uint32_t descriptorUpdateEntryCount;
    const VkDescriptorUpdateTemplateEntry*
       pDescriptorUpdateEntries;
    VkDescriptorUpdateTemplateType templateType;
    VkDescriptorSetLayout descriptorSetLayout;
    VkPipelineBindPoint pipelineBindPoint; P.13 15
    VkPipelineLayout pipelineLayout;
    uint32_t set; = 0
} VkDescriptorUpdateTemplateCreateInfo;

    VKPipelineBindPoint: VK_PIPELINE_BIND_POINT_X
       where X is GRAPHICS, COMPUTE

    templateType: VK_DESCRIPTOR_UPDATE_TEMPLATE_-
       TYPE_DESCRIPTOR_SET

typedef struct VkDescriptorUpdateTemplateEntry {
    uint32_t dstBinding;
    uint32_t dstArrayElement;
    uint32_t descriptorCount;
    VkDescriptorType descriptorType; P.12
    size_t offset; size_t stride;
} VkDescriptorUpdateTemplateEntry;

void vkDestroyDescriptorUpdateTemplate(
    VkDevice device,
    VkDescriptorUpdateTemplate
       descriptorUpdateTemplate,
    const VkAllocationCallbacks* pAllocator); P.12

void vkUpdateDescriptorSetWithTemplate(
    VkDevice device,
    VkDescriptorSet descriptorSet,
    VkDescriptorUpdateTemplate
       descriptorUpdateTemplate,
    const void* pData);
```

Descriptor Set Binding [13.2.7]

```
void vkCmdBindDescriptorSets(
    VkCommandBuffer commandBuffer,
    VkPipelineBindPoint pipelineBindPoint, P.13 15
    VkPipelineLayout layout, P.15
    uint32_t firstSet,
    uint32_t descriptorSetCount,
    const VkDescriptorSet* pDescriptorSets,
    uint32_t dynamicOffsetCount,
    const uint32_t* pDynamicOffsets);
```

Push Constant Updates [13.2.8]

The pipeline layout defines shader push constants which are updated via Vulkan commands rather than via writes to memory or copy commands.

```
void vkCmdPushConstants(
    VkCommandBuffer commandBuffer,
    VkPipelineLayout layout, P.15
    VkShaderStageFlags stageFlags, P.15
    uint32_t offset, uint32_t size,
    const void* pValues);
```

Queries [16]

Query Pools [16.1]

```
VkResult vkCreateQueryPool(
    VkDevice device,
    const VkQueryPoolCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkQueryPool* pQueryPool);

typedef struct VkQueryPoolCreateInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkQueryPoolCreateFlags flags; = 0
    VkQueryType queryType;
    uint32_t queryCount;
    VkQueryPipelineStatisticFlags pipelineStatistics; P.15
} VkQueryPoolCreateInfo;

    queryType: VK_QUERY_TYPE_OCCLUSION,
       VK_QUERY_TYPE_PIPELINE_STATISTICS,
       VK_QUERY_TYPE_TIMESTAMP

void vkDestroyQueryPool(
    VkDevice device,
    VkQueryPool queryPool,
    const VkAllocationCallbacks* pAllocator); P.12
```

Query Operation [16.2]

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

void vkCmdBeginQuery(
    VkCommandBuffer commandBuffer,
    VkQueryPool queryPool,
    uint32_t entry,
    VkQueryControlFlags flags);
    flags: VK_QUERY_CONTROL_PRECISE_BIT

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

VkResult vkGetQueryPoolResults(
    VkDevice device,
    VkQueryPool queryPool,
    uint32_t firstQuery,
    uint32_t queryCount,
    size_t dataSize,
    void* pData,
    VkDeviceSize stride,
    VkQueryResultFlags flags);
    flags: VK_QUERY_RESULT_X_BIT where X is
       64, WAIT, WITH_AVAILABILITY, PARTIAL

void vkCmdCopyQueryPoolResults(
    VkCommandBuffer commandBuffer,
    VkQueryPool queryPool,
    uint32_t firstQuery,
    uint32_t queryCount,
    VkBuffer dstBuffer,
    VkDeviceSize dstOffset,
    VkDeviceSize stride,
    VkQueryResultFlags flags);
    flags: VK_QUERY_RESULT_X_BIT where X is
       64, WAIT, WITH_AVAILABILITY, PARTIAL
```

Timestamp Queries [16.5]

```
void vkCmdWriteTimestamp(
    VkCommandBuffer commandBuffer,
    VkPipelineStageFlagBits pipelineStage, P.15
    VkQueryPool queryPool,
    uint32_t query);
```

Clear Commands [17]

Outside a Render Pass Instance [17.1]

```
void vkCmdClearColorImage(
    VkCommandBuffer commandBuffer,
    VkImage image,
    VkImageLayout imageLayout, P.13
    const VkClearColorValue* pColor, P.12
    uint32_t rangeCount,
    const VkImageSubresourceRange* pRanges); P.13
    imageLayout:
       VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
       VK_IMAGE_LAYOUT_GENERAL.
       VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR

void vkCmdClearDepthStencilImage(
    VkCommandBuffer commandBuffer,
    VkImage image,
    VkImageLayout imageLayout, P.13
    const VkClearDepthStencilValue* pDepthStencil, P.12
    uint32_t rangeCount,
    const VkImageSubresourceRange* pRanges); P.13
    imageLayout:
       VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
       VK_IMAGE_LAYOUT_GENERAL
```

Inside a Render Pass Instance [17.2]

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

typedef struct VkClearRect {
    VkRect2D rect; P.15
    uint32_t baseArrayLayer;
    uint32_t layerCount;
} VkClearRect;

typedef struct VkClearAttachment {
    VkImageAspectFlags aspectMask; P.13
    uint32_t colorAttachment;
    VkClearValue clearValue; P.12
} VkClearAttachment;
```

Filling Buffers [17.4]

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

Updating Buffers [17.5]

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

Drawing Commands [19]

```
void vkCmdBindIndexBuffer(
    VkCommandBuffer commandBuffer,
    VkBuffer buffer, VkDeviceSize offset,
    VkIndexType indexType);
  indexType: VK_INDEX_TYPE_UINT{16, 32}

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

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

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

typedef struct VkDrawIndirectCommand {
    uint32_t vertexCount; uint32_t instanceCount;
    uint32_t firstVertex; uint32_t firstInstance;
} VkDrawIndirectCommand;

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

typedef struct VkDrawIndexedIndirectCommand {
    uint32_t indexCount; uint32_t instanceCount;
    uint32_t firstIndex; int32_t vertexOffset;
    uint32_t firstInstance;
} VkDrawIndexedIndirectCommand;
```

Fixed-Function Vertex Postprocessing [23]

Controlling the Viewport [23.5]

```
void vkCmdSetViewport(
    VkCommandBuffer commandBuffer,
    uint32_t firstViewport,
    uint32_t viewportCount,
    const VkViewport* pViewports); P.15
```

Rasterization [24]

Basic Line Segment Rasterization [24.6]

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

Depth Bias [24.7.3]

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

Framebuffer: Blend Factors [26.1.1]

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

Copy Commands [18]

Copying Data Between Buffers [18.2]

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

typedef struct VkBufferCopy {
    VkDeviceSize srcOffset; VkDeviceSize dstOffset;
    VkDeviceSize size;
} VkBufferCopy;
```

Copying Data Between Images [18.3]

```
void vkCmdCopyImage(
    VkCommandBuffer commandBuffer,
    VkImage srcImage,
    VkImageLayout srcImageLayout, P.13
    VkImage dstImage,
    VkImageLayout dstImageLayout, P.13
    uint32_t regionCount,
    const VkImageCopy* pRegions);

typedef struct VkImageCopy {
    VkImageSubresourceLayers srcSubresource; P.13
    VkOffset3D srcOffset; P14
    VkImageSubresourceLayers dstSubresource; P.13
    VkOffset3D dstOffset; P.13
    VkExtent3D extent; P12
} VkImageCopy;
```

Copying Data Between Buffers and Images [18.4]

```
void vkCmdCopyBufferToImage(
    VkCommandBuffer commandBuffer,
    VkBuffer srcBuffer, VkImage dstImage,
    VkImageLayout dstImageLayout, P.13
    uint32_t regionCount,
    const VkBufferImageCopy* pRegions);

void vkCmdCopyImageToBuffer(
    VkCommandBuffer commandBuffer,
    VkImage srcImage,
    VkImageLayout srcImageLayout, P.13
    VkBuffer dstBuffer,
    uint32_t regionCount,
    const VkBufferImageCopy* pRegions);

typedef struct VkBufferImageCopy {
    VkDeviceSize bufferOffset;
    uint32_t bufferRowLength;
    uint32_t bufferImageHeight;
    VkImageSubresourceLayers imageSubresource; P.13
    VkOffset3D imageOffset; P14
    VkExtent3D imageExtent; P12
} VkBufferImageCopy;
```

Image Copies With Scaling [18.5]

```
void vkCmdBlitImage(
    VkCommandBuffer commandBuffer,
    VkImage srcImage,
    VkImageLayout srcImageLayout, P.13
    VkImage dstImage,
    VkImageLayout dstImageLayout, P.13
    uint32_t regionCount,
    const VkImageBlit* pRegions,
    VkFilter filter);
  filter: VK_FILTER_NEAREST, VK_FILTER_LINEAR

typedef struct VkImageBlit {
    VkImageSubresourceLayers srcSubresource; P.13
    VkOffset3D srcOffsets[2]; P14
    VkImageSubresourceLayers dstSubresource; P.13
    VkOffset3D dstOffsets[2]; P.13
 } VkImageBlit;
```

Resolving Multisample Images [18.6]

```
void vkCmdResolveImage(
    VkCommandBuffer commandBuffer,
    VkImage srcImage,
    VkImageLayout srcImageLayout, P.13
    VkImage dstImage,
    VkImageLayout dstImageLayout, P.13
    uint32_t regionCount,
    const VkImageResolve* pRegions);

typedef struct VkImageResolve {
    VkImageSubresourceLayers srcSubresource; P.13
    VkOffset3D srcOffset; P14
    VkImageSubresourceLayers dstSubresource; P.13
    VkOffset3D dstOffset; P14
    VkExtent3D extent; P12
} VkImageResolve;
```

Vertex Input Description [20.2]

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

Fragment Operations [25]

Scissor Test [25.2]

```
void vkCmdSetScissor(
    VkCommandBuffer commandBuffer,
    uint32_t firstScissor, uint32_t scissorCount,
    const VkRect2D* pScissors); P.15
```

Depth Bounds Test [25.8]

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

Stencil Test [25.9]

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

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

void vkCmdSetStencilReference(
    VkCommandBuffer commandBuffer,
    VkStencilFaceFlags faceMask,
    uint32_t reference);
  faceMask: VK_STENCIL_FACE_{FRONT, BACK}_BIT,
    VK_STENCIL_FRONT_AND_BACK
```

Dispatching Commands [27]

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

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

typedef struct VkDispatchIndirectCommand {
    uint32_t x;
    uint32_t y;
    uint32_t z;
} VkDispatchIndirectCommand;

void vkCmdDispatchBase(
    VkCommandBuffer commandBuffer,
    uint32_t baseGroupX, uint32_t baseGroupY,
    uint32_t baseGroupZ, uint32_t groupCountX,
    uint32_t groupCountY, uint32_t groupCountZ);
```

Sparse Resources [28]

Sparse Image Format Properties [28.7.3]

```
void vkGetPhysicalDeviceSparseImageFormatProperties(
    VkPhysicalDevice physicalDevice, VkFormat format, P.13
    VkImageType type, P.13
    VkSampleCountFlagBits samples, P.15
    VkImageUsageFlags usage, P.13
    VkImageTiling tiling, P.13
    uint32_t* pPropertyCount,
    VkSparseImageFormatProperties* pProperties);

typedef struct VkSparseImageFormatProperties {
    VkImageAspectFlags aspectMask; P.13
    VkExtent3D imageGranularity; P12
    VkSparseImageFormatFlags flags;
} VkSparseImageFormatProperties;
  flags: VK_SPARSE_IMAGE_FORMAT_X where X is
    SINGLE_MIPTAIL_BIT, ALIGNED_MIP_SIZE_BIT,
    NONSTANDARD_BLOCK_SIZE_BIT

void vkGetPhysicalDeviceSparseImageFormatProperties2(
    VkPhysicalDevice physicalDevice,
    const VkPhysicalDeviceSparseImageFormatInfo2*
        pFormatInfo,
    uint32_t* pPropertyCount,
    VkSparseImageFormatProperties2* pProperties);

typedef struct VkSparseImageFormatProperties2 {
    VkStructureType sType; P.15
    void* pNext;
    VkSparseImageFormatProperties properties;
} VkSparseImageFormatProperties2;

typedef struct VkPhysicalDeviceSparseImageFormatInfo2 {
    VkStructureType sType; P.15
    const void* pNext;
    VkFormat format, P.13
    VkImageType type, P.13
    VkSampleCountFlagBits samples, P.15
    VkImageUsageFlags usage, P.13
    VkImageTiling tiling, P.13
} VkPhysicalDeviceSparseImageFormatInfo2;
```

Sparse Resource Memory Requirements [28.7.5]

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

Continued on next page >

Sparse Resources (continued)

```
typedef struct VkSparseImageMemoryRequirements {
    VkSparseImageFormatProperties formatProperties;
    uint32_t imageMipTailFirstLod;
    VkDeviceSize imageMipTailSize;
    VkDeviceSize imageMipTailOffset;
    VkDeviceSize imageMipTailStride;
} VkSparseImageMemoryRequirements;
```

```
void vkGetImageSparseMemoryRequirements2(
    VkDevice device,
    const VkImageSparseMemoryRequirementsInfo2* pInfo,
    uint32_t* pSparseMemoryRequirementCount,
    VkSparseImageMemoryRequirements2*
        pSparseMemoryRequirements);
```

```
typedef struct VkImageSparseMemoryRequirementsInfo2 {
    VkStructureType sType; P.15
    const void* pNext;
    VkImage image;
} VkImageSparseMemoryRequirementsInfo2;
```

```
typedef struct VkSparseImageMemoryRequirements2 {
    VkStructureType sType; P.15
    void* pNext;
    VkSparseImageMemoryRequirements
        memoryRequirements;
} VkSparseImageMemoryRequirements2;
```

Binding Resource Memory [28.7.6]

```
typedef struct VkBindSparseInfo {
    VkStructureType sType; P.15
    const void* pNext;
    uint32_t waitSemaphoreCount;
    const VkSemaphore* pWaitSemaphores;
    uint32_t bufferBindCount;
    const VkSparseBufferMemoryBindInfo* pBufferBinds;
    uint32_t imageOpaqueBindCount;
    const VkSparseImageOpaqueMemoryBindInfo*
        pImageOpaqueBinds;
    uint32_t imageBindCount;
    const VkSparseImageMemoryBindInfo* pImageBinds;
    uint32_t signalSemaphoreCount;
    const VkSemaphore* pSignalSemaphores;
} VkBindSparseInfo;
```

pNext must be NULL or point to:
VkDeviceGroupBindSparseInfo P.12

```
typedef struct VkSparseBufferMemoryBindInfo {
    VkBuffer buffer;
    uint32_t bindCount;
    const VkSparseMemoryBind* pBinds; P.15
} VkSparseBufferMemoryBindInfo;
```

```
typedef struct VkSparseImageOpaqueMemoryBindInfo {
    VkImage image;
    uint32_t bindCount;
    const VkSparseMemoryBind* pBinds; P.15
} VkSparseImageOpaqueMemoryBindInfo;
```

```
typedef struct VkSparseImageMemoryBindInfo {
    VkImage image;
    uint32_t bindCount;
    const VkSparseImageMemoryBind* pBinds;
} VkSparseImageMemoryBindInfo;
```

```
typedef struct VkSparseImageMemoryBind {
    VkImageSubresource subresource;
    VkOffset3D offset; P14
    VkExtent3D extent; P12
    VkDeviceMemory memory;
    VkDeviceSize memoryOffset;
    VkSparseMemoryBindFlags flags;
} VkSparseImageMemoryBind;
```

flags: VK_SPARSE_MEMORY_BIND_METADATA_BIT

```
VkResult vkQueueBindSparse(
    VkQueue queue,
    uint32_t bindInfoCount,
    const VkBindSparseInfo* pBindInfo,
    VkFence fence);
```

Window System Integration (WSI) [29]

Android Platform [29.2.1]

```
VkResult vkCreateAndroidSurfaceKHR(
    VkInstance instance,
    const VkAndroidSurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkSurfaceKHR* pSurface);
```

```
typedef struct VkAndroidSurfaceCreateInfoKHR {
    VkStructureType sType; P.15
    const void* pNext;
    VkAndroidSurfaceCreateFlagsKHR flags; = 0
    struct ANativeWindow* window;
} VkAndroidSurfaceCreateInfoKHR;
```

Wayland Platform [29.2.3]

```
VkResult vkCreateWaylandSurfaceKHR(
    VkInstance instance,
    const VkWaylandSurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkSurfaceKHR* pSurface);
```

```
typedef struct VkWaylandSurfaceCreateInfoKHR {
    VkStructureType sType; P.15
    const void* pNext;
    VkWaylandSurfaceCreateFlagsKHR flags; = 0
    struct wl_display* display;
    struct wl_surface* surface;
} VkWaylandSurfaceCreateInfoKHR;
```

Win32 Platform [29.2.4]

```
VkResult vkCreateWin32SurfaceKHR(
    VkInstance instance,
    const VkWin32SurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkSurfaceKHR* pSurface);
```

```
typedef struct VkWin32SurfaceCreateInfoKHR {
    VkStructureType sType; P.15
    const void* pNext;
    VkWin32SurfaceCreateFlagsKHR flags; = 0
    HINSTANCE hinstance; HWND hwnd;
} VkWin32SurfaceCreateInfoKHR;
```

XCB Platform [29.2.5]

```
VkResult vkCreateXcbSurfaceKHR(
    VkInstance instance,
    const VkXcbSurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkSurfaceKHR* pSurface);
```

```
typedef struct VkXcbSurfaceCreateInfoKHR {
    VkStructureType sType; P.15
    const void* pNext;
    VkXcbSurfaceCreateFlagsKHR flags; = 0
    xcb_connection_t* connection; xcb_window_t window;
} VkXcbSurfaceCreateInfoKHR;
```

Xlib Platform [29.2.6]

```
VkResult vkCreateXlibSurfaceKHR(
    VkInstance instance,
    const VkXlibSurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkSurfaceKHR* pSurface);
```

```
typedef struct VkXlibSurfaceCreateInfoKHR {
    VkStructureType sType; P.15
    const void* pNext;
    VkXlibSurfaceCreateFlagsKHR flags; = 0
    Display* dpy; Window window;
} VkXlibSurfaceCreateInfoKHR;
```

Platform-Independent Information [29.2.7]

```
void vkDestroySurfaceKHR(
    VkInstance instance, VkSurfaceKHR surface,
    const VkAllocationCallbacks* pAllocator); P.12
```

Display Enumeration [29.3.1]

```
VkResult vkGetPhysicalDeviceDisplayPropertiesKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t* pPropertyCount,
    VkDisplayPropertiesKHR* pProperties);
```

```
typedef struct VkDisplayPropertiesKHR {
    VkDisplayKHR display; const char* displayName;
    VkExtent2D physicalDimensions; P12
    VkExtent2D physicalResolution; P.12
    VkSurfaceTransformFlagsKHR supportedTransforms; P.15
    VkBool32 planeReorderPossible;
    VkBool32 persistentContent;
} VkDisplayPropertiesKHR;
```

```
VkResult vkGetPhysicalDeviceDisplayProperties2KHR (
    VkPhysicalDevice physicalDevice,
    uint32_t* pPropertyCount,
    VkDisplayProperties2KHR* pProperties);
```

```
typedef struct VkDisplayProperties2KHR {
    VkStructureType sType; P.15
    void* pNext;
    VkDisplayPropertiesKHR displayProperties;
} VkDisplayProperties2KHR;
```

Display Planes

```
VkResult vkGetPhysicalDeviceDisplayPlanePropertiesKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t* pPropertyCount,
    VkDisplayPlanePropertiesKHR* pProperties);
```

```
typedef struct VkDisplayPlanePropertiesKHR {
    VkDisplayKHR currentDisplay; uint32_t currentStackIndex;
} VkDisplayPlanePropertiesKHR;
```

```
VkResult vkGetPhysicalDeviceDisplayPlaneProperties2KHR (
    VkPhysicalDevice physicalDevice,
    uint32_t* pPropertyCount,
    VkDisplayPlaneProperties2KHR* pProperties);
```

```
typedef struct VkDisplayPlaneProperties2KHR {
    VkStructureType sType; P.15
    void* pNext;
    VkDisplayPlanePropertiesKHR displayPlaneProperties;
} VkDisplayPlaneProperties2KHR;
```

```
VkResult vkGetDisplayPlaneSupportedDisplaysKHR(
    VkPhysicalDevice physicalDevice, uint32_t planeIndex,
    uint32_t* pDisplayCount, VkDisplayKHR* pDisplays);
```

Display Modes

```
VkResult vkGetDisplayModePropertiesKHR(
    VkPhysicalDevice physicalDevice, VkDisplayKHR display,
    uint32_t* pPropertyCount,
    VkDisplayModePropertiesKHR* pProperties);
```

```
typedef struct VkDisplayModePropertiesKHR {
    VkDisplayModeKHR displayMode;
    VkDisplayModeParametersKHR parameters;
} VkDisplayModePropertiesKHR;
```

```
typedef struct VkDisplayModeParametersKHR {
    VkExtent2D visibleRegion; P12
    uint32_t refreshRate;
} VkDisplayModeParametersKHR;
```

```
VkResult vkGetDisplayModeProperties2KHR (
    VkPhysicalDevice physicalDevice,
    VkDisplayKHR display, uint32_t* pPropertyCount,
    VkDisplayModeProperties2KHR* pProperties);
```

```
typedef struct VkDisplayModeProperties2KHR {
    VkStructureType sType; P.15
    void* pNext;
    VkDisplayModePropertiesKHR displayModeProperties;
} VkDisplayModeProperties2KHR;
```

```
VkResult vkCreateDisplayModeKHR(
    VkPhysicalDevice physicalDevice, VkDisplayKHR display,
    const VkDisplayModeCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkDisplayModeKHR* pMode);
```

```
typedef struct VkDisplayModeCreateInfoKHR {
    VkStructureType sType; P.15
    const void* pNext;
    VkDisplayModeCreateFlagsKHR flags; = 0
    VkDisplayModeParametersKHR parameters;
} VkDisplayModeCreateInfoKHR;
```

```
VkResult vkGetDisplayPlaneCapabilitiesKHR(
    VkPhysicalDevice physicalDevice,
    VkDisplayModeKHR mode, uint32_t planeIndex,
    VkDisplayPlaneCapabilitiesKHR* pCapabilities);
```

```
typedef struct VkDisplayPlaneCapabilitiesKHR {
    VkDisplayPlaneAlphaFlagsKHR supportedAlpha;
    VkOffset2D minSrcPosition; P14
    VkOffset2D maxSrcPosition; P14
    VkExtent2D minSrcExtent; P12
    VkExtent2D maxSrcExtent; P12
    VkOffset2D minDstPosition; P14
    VkOffset2D maxDstPosition; P14
    VkExtent2D minDstExtent; P12
    VkExtent2D maxDstExtent; P12
} VkDisplayPlaneCapabilitiesKHR;
```

```
VkResult vkGetDisplayPlaneCapabilities2KHR (
    VkPhysicalDevice physicalDevice,
    const VkDisplayPlaneInfo2KHR* pDisplayPlaneInfo,
    VkDisplayPlaneCapabilities2KHR* pCapabilities);
```

```
typedef struct VkDisplayPlaneInfo2KHR {
    VkStructureType sType; P.15
    const void* pNext;
    VkDisplayModeKHR mode; uint32_t planeIndex;
} VkDisplayPlaneInfo2KHR;
```

```
typedef struct VkDisplayPlaneCapabilities2KHR {
    VkStructureType sType; P.15
    void* pNext;
    VkDisplayPlaneCapabilitiesKHR capabilities;
} VkDisplayPlaneCapabilities2KHR;
```

Continued on next page >

WSI (continued)

Display Surfaces [29.3.2]

```
VkResult vkCreateDisplayPlaneSurfaceKHR(
    VkInstance instance,
    const VkDisplaySurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkSurfaceKHR* pSurface);
```

```
typedef struct VkDisplaySurfaceCreateInfoKHR {
    VkStructureType sType; P.15
    const void* pNext;
    VkDisplaySurfaceCreateFlagsKHR flags; = 0
    VkDisplayModeKHR displayMode;
    uint32_t planeIndex; uint32_t planeStackIndex;
    VkSurfaceTransformFlagBitsKHR transform; P.15
    float globalAlpha;
    VkDisplayPlaneAlphaFlagBitsKHR alphaMode;
    VkExtent2D imageExtent; P.12
} VkDisplaySurfaceCreateInfoKHR;
```

alphaMode: VK_DISPLAY_PLANE_ALPHA_*X*_BIT_KHR
where *X* is OPAQUE, GLOBAL, PER_PIXEL, PER_PIXEL_PREMULTIPLIED

Querying for WSI Support [29.4]

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

Wayland Platform Querying [29.4.3]

```
VkBool32
    vkGetPhysicalDeviceWaylandPresentationSupportKHR(
        VkPhysicalDevice physicalDevice,
        uint32_t queueFamilyIndex,
        struct wl_display* display);
```

Win32 Platform Querying [29.4.4]

```
VkBool32
    vkGetPhysicalDeviceWin32PresentationSupportKHR(
        VkPhysicalDevice physicalDevice,
        uint32_t queueFamilyIndex);
```

XCB Platform Querying [29.4.5]

```
VkBool32
    vkGetPhysicalDeviceXcbPresentationSupportKHR(
        VkPhysicalDevice physicalDevice,
        uint32_t queueFamilyIndex,
        xcb_connection_t* connection,
        xcb_visualid_t visual_id);
```

Xlib Platform Querying [29.4.6]

```
VkBool32
    vkGetPhysicalDeviceXlibPresentationSupportKHR(
        VkPhysicalDevice physicalDevice,
        uint32_t queueFamilyIndex,
        Display* dpy, VisualID visualID);
```

Surface Queries [29.5]

```
VkResult vkGetPhysicalDeviceSurfaceCapabilitiesKHR(
    VkPhysicalDevice physicalDevice, VkSurfaceKHR surface,
    VkSurfaceCapabilitiesKHR* pSurfaceCapabilities); P.15
```

```
VkResult vkGetPhysicalDeviceSurfaceCapabilities2KHR(
    VkPhysicalDevice physicalDevice,
    const VkPhysicalDeviceSurfaceInfo2KHR* pSurfaceInfo, P.15
    VkSurfaceCapabilities2KHR* pSurfaceCapabilities);
```

```
typedef struct VkSurfaceCapabilities2KHR {
    VkStructureType sType; P.15
    void* pNext;
    VkSurfaceCapabilitiesKHR surfaceCapabilities; P.15
} VkSurfaceCapabilities2KHR;
```

pNext must be NULL or point to:
VkSharedPresentSurfaceCapabilitiesKHR

```
typedef struct VkSharedPresentSurfaceCapabilitiesKHR {
    VkStructureType sType; P.15
    void* pNext;
    VkImageUsageFlags
        sharedPresentSupportedUsageFlags; P.13
} VkSharedPresentSurfaceCapabilitiesKHR;
```

```
VkResult vkGetPhysicalDeviceSurfaceFormatsKHR(
    VkPhysicalDevice physicalDevice,
    VkSurfaceKHR surface,
    uint32_t* pSurfaceFormatCount,
    VkSurfaceFormatKHR* pSurfaceFormats); P.15
```

```
VkResult vkGetPhysicalDeviceSurfaceFormats2KHR(
    VkPhysicalDevice physicalDevice,
    const VkPhysicalDeviceSurfaceInfo2KHR* pSurfaceInfo, P.15
    uint32_t* pSurfaceFormatCount,
    VkSurfaceFormat2KHR* pSurfaceFormats);
```

```
typedef struct VkSurfaceFormat2KHR {
    VkStructureType sType; P.15
    void* pNext;
    VkSurfaceFormatKHR surfaceFormat; P.15
} VkSurfaceFormat2KHR;
```

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

pPresentModes: VK_PRESENT_MODE_*X*_KHR
where *X* is IMMEDIATE, MAILBOX, FIFO, FIFO_RELAXED, SHARED_DEMAND_REFRESH, SHARED_CONTINUOUS_REFRESH

Device Group Queries [29.6]

```
VkResult vkGetDeviceGroupPresentCapabilitiesKHR(
    VkDevice device,
    VkDeviceGroupPresentCapabilitiesKHR*
        pDeviceGroupPresentCapabilities);
```

```
typedef struct VkDeviceGroupPresentCapabilitiesKHR {
    VkStructureType sType; P.15
    const void* pNext;
    uint32_t presentMask[VK_MAX_DEVICE_GROUP_SIZE];
    VkDeviceGroupPresentModeFlagsKHR modes; P.12
} VkDeviceGroupPresentCapabilitiesKHR;
```

```
VkResult vkGetDeviceGroupSurfacePresentModesKHR(
    VkDevice device, VkSurfaceKHR surface,
    VkDeviceGroupPresentModeFlagsKHR* pModes); P.12
```

```
VkResult vkGetPhysicalDevicePresentRectanglesKHR(
    VkPhysicalDevice physicalDevice,
    VkSurfaceKHR surface, uint32_t* pRectCount,
    VkRect2D* pRects); P.15
```

WSI Swapchain [29.7]

```
VkResult vkGetSwapchainStatusKHR(
    VkDevice device,
    VkSwapchainKHR swapchain);
```

```
VkResult vkCreateSwapchainKHR(
    VkDevice device,
    const VkSwapchainCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.12
    VkSwapchainKHR* pSwapchain);
```

```
typedef struct VkSwapchainCreateInfoKHR {
    VkStructureType sType; P.15
    const void* pNext;
    VkSwapchainCreateFlagsKHR flags;
    VkSurfaceKHR surface; uint32_t minImageCount;
    VkFormat imageFormat; P.13
    VkColorSpaceKHR imageColorSpace;
    VkExtent2D imageExtent; P.12
    uint32_t imageArrayLayers;
    VkImageUsageFlags imageUsage; P.13
    VkSharingMode imageSharingMode; P.15
    uint32_t queueFamilyIndexCount;
    const uint32_t* pQueueFamilyIndices;
    VkSurfaceTransformFlagBitsKHR preTransform; P.15
    VkCompositeAlphaFlagBitsKHR compositeAlpha; P.12
    VkPresentModeKHR presentMode;
    VkBool32 clipped; VkSwapchainKHR oldSwapchain;
} VkSwapchainCreateInfoKHR;
```

pNext: may point to struct:
VkDeviceGroupSwapchainCreateInfoKHR

flags: VK_SWAPCHAIN_CREATE_*X*_KHR where *X* is SPLIT_INSTANCE_BIND_REGIONS, PROTECTED

colorSpace: VK_COLOR_SPACE_SRGB_NONLINEAR_KHR

presentMode: VK_PRESENT_MODE_*X*_KHR
where *X* is IMMEDIATE, MAILBOX, FIFO, FIFO_RELAXED, DEMAND_REFRESH, CONTINUOUS_REFRESH

```
typedef struct VkDeviceGroupSwapchainCreateInfoKHR {
    VkStructureType sType; P.15
    const void* pNext;
    VkDeviceGroupPresentModeFlagsKHR modes;
} VkDeviceGroupSwapchainCreateInfoKHR;
```

modes: VK_DEVICE_GROUP_PRESENT_MODE_*X*_BIT_KHR
where *X* is LOCAL, REMOTE, SUM, LOCAL_MULTI_DEVICE

```
void vkDestroySwapchainKHR(
    VkDevice device, VkSwapchainKHR swapchain,
    const VkAllocationCallbacks* pAllocator); P.12
```

```
VkResult vkCreateSharedSwapchainsKHR(
    VkDevice device,
    uint32_t swapchainCount,
    const VkSwapchainCreateInfoKHR* pCreateInfos,
    const VkAllocationCallbacks* pAllocator, P.12
    VkSwapchainKHR* pSwapchains);
```

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

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

```
VkResult vkAcquireNextImage2KHR(
    VkDevice device,
    const VkAcquireNextImageInfoKHR* pAcquireInfo,
    uint32_t* pImageIndex);
```

```
typedef struct VkAcquireNextImageInfoKHR {
    VkStructureType sType; P.15 const void* pNext;
    VkSwapchainKHR swapchain; uint64_t timeout;
    VkSemaphore semaphore;
    VkFence fence; uint32_t deviceMask;
} VkAcquireNextImageInfoKHR;
```

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

```
typedef struct VkPresentInfoKHR {
    VkStructureType sType; P.15
    const void* pNext; uint32_t waitSemaphoreCount;
    const VkSemaphore* pWaitSemaphores;
    uint32_t swapchainCount;
    const VkSwapchainKHR* pSwapchains;
    const uint32_t* pImageIndices; VkResult* pResults;
} VkPresentInfoKHR;
```

pNext must be NULL or point to one of:
VkDeviceGroupPresentInfoKHR, VkDisplayPresentInfoKHR, or VkPresentRegionsKHR

```
typedef struct VkDeviceGroupPresentInfoKHR {
    VkStructureType sType; P.15
    const void* pNext; uint32_t swapchainCount;
    const uint32_t* pDeviceMasks;
    VkDeviceGroupPresentModeFlagBitsKHR mode;
} VkDeviceGroupPresentInfoKHR;
```

mode: VK_DEVICE_GROUP_PRESENT_MODE_*X*_BIT_KHR
where *X* is REMOTE, SUM, LOCAL, LOCAL_MULTI_DEVICE

```
typedef struct VkDisplayPresentInfoKHR {
    VkStructureType sType; P.15 const void* pNext;
    VkRect2D srcRect; P.15
    VkRect2D dstRect; P.15
    VkBool32 persistent;
} VkDisplayPresentInfoKHR;
```

```
typedef struct VkPresentRegionsKHR {
    VkStructureType sType; P.15
    const void* pNext; uint32_t swapchainCount;
    const VkPresentRegionKHR* pRegions;
} VkPresentRegionsKHR;
```

```
typedef struct VkPresentRegionKHR {
    uint32_t rectangleCount;
    const VkRectLayerKHR* pRectangles;
} VkPresentRegionKHR;
```

```
typedef struct VkRectLayerKHR {
    VkOffset2D offset; P.14
    VkExtent2D extent; P.12
    uint32_t layer;
} VkRectLayerKHR;
```

```
typedef struct VkDisplayPresentInfoKHR {
    VkStructureType sType; P.15 const void* pNext;
    VkRect2D srcRect; P.15
    VkRect2D dstRect; P.15
    VkBool32 persistent;
} VkDisplayPresentInfoKHR;
```

Extended Functionality

Layers [30.1]

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

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

```
typedef struct VkLayerProperties {
    char layerName [VK_MAX_EXTENSION_NAME_SIZE];
    uint32_t specVersion;
    uint32_t implementationVersion;
    char description [VK_MAX_DESCRIPTION_SIZE];
} VkLayerProperties;
```

Extensions [30.2]

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

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

Continued on next page >

Extended Functionality (continued)

```
typedef struct VkExtensionProperties {
    char extensionName [VK_MAX_EXTENSION_NAME_SIZE];
    uint32_t specVersion;
} VkExtensionProperties;
```

Additional Buffer Capabilities [31.5]

```
void vkGetPhysicalDeviceExternalBufferProperties(
    VkPhysicalDevice physicalDevice,
    const VkPhysicalDeviceExternalBufferInfo*
        pExternalBufferInfo,
    VkExternalBufferProperties*
        pExternalBufferProperties);

typedef struct VkPhysicalDeviceExternalBufferInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkBufferCreateFlags flags; P.12
    VkBufferUsageFlags usage; P.12
    VkExternalMemoryHandleTypeFlagBits handleType; P.12
} VkPhysicalDeviceExternalBufferInfo;

typedef struct VkExternalBufferProperties {
    VkStructureType sType; P.15
    void* pNext;
    VkExternalMemoryProperties
        externalMemoryProperties; P.13
} VkExternalBufferProperties;
```

Optional Semaphore Capabilities [31.6]

```
void vkGetPhysicalDeviceExternalSemaphoreProperties(
    VkPhysicalDevice physicalDevice,
    const VkPhysicalDeviceExternalSemaphoreInfo*
        pExternalSemaphoreInfo,
    VkExternalSemaphoreProperties*
        pExternalSemaphoreProperties);

typedef struct VkPhysicalDeviceExternalSemaphoreInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkExternalSemaphoreHandleTypeFlagBits
        handleType; P.13
} VkPhysicalDeviceExternalSemaphoreInfo;
```

enum VkExternalSemaphoreHandleTypeFlagBits:
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_*X*_BIT where *X* is OPAQUE_FD, OPAQUE_WIN32[_KMT], D3D12_FENCE, SYNC_FD

```
typedef struct VkExternalSemaphoreProperties {
    VkStructureType sType; P.15
    void* pNext;
    VkExternalSemaphoreHandleTypeFlags
        exportFromImportedHandleTypes; P.13
    VkExternalSemaphoreHandleTypeFlags
        compatibleHandleTypes;
    VkExternalSemaphoreFeatureFlags
        externalSemaphoreFeatures;
} VkExternalSemaphoreProperties;
```

VkExternalSemaphoreFeatureFlagBits:
VK_EXTERNAL_SEMAPHORE_FEATURE_*X*_BIT where *X* is EXPORTABLE, IMPORTABLE

Optional Fence Capabilities [31.7]

```
void vkGetPhysicalDeviceExternalFenceProperties(
    VkPhysicalDevice physicalDevice,
    const VkPhysicalDeviceExternalFenceInfo*
        pExternalFenceInfo,
    VkExternalFenceProperties* pExternalFenceProperties);

typedef struct VkPhysicalDeviceExternalFenceInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkExternalFenceHandleTypeFlagBits handleType; P.12
} VkPhysicalDeviceExternalFenceInfo;

typedef struct VkExternalFenceProperties {
    VkStructureType sType; P.15
    void* pNext;
    VkExternalFenceHandleTypeFlags
        exportFromImportedHandleTypes; P.12
    VkExternalFenceHandleTypeFlags
        compatibleHandleTypes; P.12
    VkExternalFenceFeatureFlags externalFenceFeatures;
} VkExternalFenceProperties;
```

enum VkExternalFenceFeatureFlagBits:
VK_EXTERNAL_FENCE_FEATURE_*X*_BIT where *X* is EXPORTABLE, IMPORTABLE

Features, Limits, and Formats [31]

Features [31.1]

```
void vkGetPhysicalDeviceFeatures(
    VkPhysicalDevice physicalDevice,
    VkPhysicalDeviceFeatures* pFeatures); P.14

void vkGetPhysicalDeviceFeatures2(
    VkPhysicalDevice physicalDevice,
    VkPhysicalDeviceFeatures2* pFeatures); P.14
```

Format Properties [31.3.2]

```
void vkGetPhysicalDeviceFormatProperties(
    VkPhysicalDevice physicalDevice,
    VkFormat format, P.13
    VkFormatProperties* pFormatProperties);

typedef struct VkFormatProperties {
    VkFormatFeatureFlags linearTilingFeatures;
    VkFormatFeatureFlags optimalTilingFeatures;
    VkFormatFeatureFlags bufferFeatures;
} VkFormatProperties;
```

enum VkFormatFeatureFlagBits:
VK_FORMAT_FEATURE_*X*_BIT where *X* is SAMPLED_IMAGE, STORAGE_IMAGE[_ATOMIC], UNIFORM_TEXEL_BUFFER, STORAGE_TEXEL_BUFFER[_ATOMIC], VERTEX_BUFFER, COLOR_ATTACHMENT[_BLEND], DEPTH_STENCIL_ATTACHMENT, SAMPLED_IMAGE_FILTER_LINEAR, DISJOINT, BLIT_{SRC, DST}, TRANSFER_{SRC, DST}, {MIDPOINT, COSITED}_CHROMA_SAMPLES,
and VK_FORMAT_FEATURE_SAMPLED_IMAGE_YCBCR_-CONVERSION_*X* where *X* is LINEAR_FILTER, SEPARATE_RECONSTRUCTION_FILTER, CHROMA_RECONSTRUCTION_EXPLICIT, CHROMA_RECONSTRUCTION_EXPLICIT_FORCEABLE

```
void vkGetPhysicalDeviceFormatProperties2(
    VkPhysicalDevice physicalDevice,
    VkFormat format, P.13
    VkFormatProperties2* pFormatProperties);

typedef struct VkFormatProperties2 {
    VkStructureType sType; P.15
    void* pNext;
    VkFormatProperties formatProperties;
} VkFormatProperties2;
```

Additional Image Capabilities [31.4]

```
VkResult vkGetPhysicalDeviceImageFormatProperties(
    VkPhysicalDevice physicalDevice,
    VkFormat format, P.13
    VkImageType type, P.13
    VkImageTiling tiling, P.13
    VkImageUsageFlags usage, P.13
    VkImageCreateFlags flags, P.13
    VkImageFormatProperties* pImageFormatProperties);

typedef struct VkImageFormatProperties {
    VkExtent3D maxExtent; P.12
    uint32_t maxMipLevels;
    uint32_t maxArrayLayers;
    VkSampleCountFlags sampleCounts; P.15
    VkDeviceSize maxResourceSize;
} VkImageFormatProperties;

VkResult vkGetPhysicalDeviceImageFormatProperties2(
    VkPhysicalDevice physicalDevice,
    const VkPhysicalDeviceImageFormatInfo2*
        pImageFormatInfo,
    VkImageFormatProperties2* pImageFormatProperties);

typedef struct VkImageFormatProperties2 {
    VkStructureType sType; P.15
    void* pNext;
    VkImageFormatProperties imageFormatProperties;
} VkImageFormatProperties2;
```

pNext must be NULL or point to:
VkExternalImageFormatProperties P.12
VkSamplerYcbcrConversionImageFormatProperties P.15

```
typedef struct VkPhysicalDeviceImageFormatInfo2 {
    VkStructureType sType; P.15
    const void* pNext;
    VkFormat format, P.13
    VkImageType type, P.13
    VkImageTiling tiling, P.13
    VkImageUsageFlags usage, P.13
    VkImageCreateFlags flags, P.13
} VkPhysicalDeviceImageFormatInfo2;
```

pNext must be NULL or point to:
VkPhysicalDeviceExternalImageFormatInfo P.14

Notes

Structures and Enumerations

This section contains an alphabetic reference to types enums and structs referenced in multiple places on preceding pages.

enum **VkAccessFlagBits**:

```
  VK_ACCESS_X_BIT where X is
  INDIRECT_COMMAND_READ,
  INDEX_READ,
  VERTEX_ATTRIBUTE_READ,
  UNIFORM_READ,
  INPUT_ATTACHMENT_READ,
  SHADER_[READ, WRITE],
  COLOR_ATTACHMENT_[READ, WRITE],
  DEPTH_STENCIL_ATTACHMENT_[READ, WRITE],
  TRANSFER_[READ, WRITE],
  HOST_[READ, WRITE],
  MEMORY_[READ, WRITE]
```

```
typedef struct VkAllocationCallbacks {
    void* pUserData;
    PFN_vkAllocationFunction pfnAllocation;
    PFN_vkReallocationFunction pfnReallocation;
    PFN_vkFreeFunction pfnFree;
    PFN_vkInternalAllocationNotification
        pfnInternalAllocation;
    PFN_vkInternalFreeNotification pfnInternalFree;
} VkAllocationCallbacks;
```

```
typedef void* (VKAPI_PTR* PFN_vkAllocationFunction)(
    void* pUserData,
    size_t size,
    size_t alignment,
    VkSystemAllocationScope allocationScope);
```

```
typedef void* (VKAPI_PTR* PFN_vkReallocationFunction)(
    void* pUserData,
    void* pOriginal,
    size_t size,
    size_t alignment,
    VkSystemAllocationScope allocationScope);
```

```
typedef void (VKAPI_PTR*  PFN_vkFreeFunction)(
    void* pUserData,
    void* pMemory);
```

```
typedef void (
    VKAPI_PTR*  PFN_vkInternalAllocationNotification)(
        void* pUserData,
        size_t size,
        VkInternalAllocationType allocationType,
        VkSystemAllocationScope allocationScope);
```

```
typedef void (
    VKAPI_PTR* PFN_vkInternalFreeNotification)(
        void* pUserData,
        size_t size,
        VkInternalAllocationType allocationType,
        VkSystemAllocationScope allocationScope);
```

allocationType:
VK_INTERNAL_ALLOCATION_TYPE_EXECUTABLE

allocationScope: VK_SYSTEM_ALLOCATION_SCOPE_*X* where *X* is COMMAND, OBJECT, CACHE, DEVICE, INSTANCE

```
typedef struct VkBindBufferMemoryDeviceGroupInfo {
    VkStructureType sType;  P.15
    const void* pNext; uint32_t deviceIndexCount;
    const uint32_t* pDeviceIndices;
} VkBindBufferMemoryDeviceGroupInfo;
```

```
typedef struct VkBindImageMemoryDeviceGroupInfo {
    VkStructureType sType;  P.15
    const void* pNext; uint32_t deviceIndexCount;
    const uint32_t* pDeviceIndices;
    uint32_t splitInstanceBindRegionCount;
    const VkRect2D* pSplitInstanceBindRegions;  P.15
} VkBindImageMemoryDeviceGroupInfo;
```

```
typedef struct VkBindImagePlaneMemoryInfo {
    VkStructureType sType;  P.15
    const void* pNext;
    VkImageAspectFlagBits planeAspect;  P.13
} VkBindImagePlaneMemoryInfo;
```

enum **VkBlendOp**:

```
  VK_BLEND_OP_ADD,
  VK_BLEND_OP_SUBTRACT,
  VK_BLEND_OP_REVERSE_SUBTRACT,
  VK_BLEND_OP_MIN,
  VK_BLEND_OP_MAX
```

enum **VkBufferCreateFlagBits**:

```
  VK_BUFFER_CREATE_SPARSE_BINDING_BIT,
  VK_BUFFER_CREATE_SPARSE_RESIDENCY_BIT,
  VK_BUFFER_CREATE_SPARSE_ALIASED_BIT,
  VK_BUFFER_CREATE_PROTECTED_BIT
```

```
typedef struct VkBufferMemoryBarrier {
    VkStructureType sType;  P.15
    const void* pNext;
    VkAccessFlags srcAccessMask;  P.12
    VkAccessFlags dstAccessMask;  P.12
    uint32_t srcQueueFamilyIndex;
    uint32_t dstQueueFamilyIndex;
    VkBuffer buffer;
    VkDeviceSize offset; VkDeviceSize size;
} VkBufferMemoryBarrier;
```

enum **VkBufferUsageFlagBits**:

```
  VK_BUFFER_USAGE_X_BIT where X is
  TRANSFER_SRC, TRANSFER_DST,
  UNIFORM_TEXEL_BUFFER, STORAGE_TEXEL_BUFFER,
  UNIFORM_BUFFER, STORAGE_BUFFER, INDEX_BUFFER,
  VERTEX_BUFFER, INDIRECT_BUFFER
```

```
typedef union VkClearColorValue {
    float float32[4];
    int32_t int32[4];
    uint32_t uint32[4];
} VkClearColorValue;
```

```
typedef struct VkClearDepthStencilValue {
    float depth;
    uint32_t stencil;
} VkClearDepthStencilValue;
```

```
typedef union VkClearValue {
    VkClearColorValue color;  P.12
    VkClearDepthStencilValue depthStencil;  P.12
} VkClearValue;
```

enum **VkCompareOp:**

```
  VK_COMPARE_OP_X where X is
  NEVER, LESS, EQUAL,
  LESS_OR_EQUAL,
  GREATER,
  NOT_EQUAL,
  GREATER_OR_EQUAL,
  ALWAYS
```

enum **VkCompositeAlphaFlagBitsKHR**:

```
  VK_COMPOSITE_ALPHA_X_BIT_KHR where X is
  OPAQUE, PRE_MULTIPLIED, POST_MULTIPLIED, INHERIT
```

enum **VkDependencyFlagBits**:

```
  VK_DEPENDENCY_BY_REGION_BIT,
  VK_DEPENDENCY_DEVICE_GROUP_BIT,
  VK_DEPENDENCY_VIEW_LOCAL_BIT
```

enum **VkDescriptorType**:

```
  VK_DESCRIPTOR_TYPE_X where X is
  SAMPLER, COMBINED_IMAGE_SAMPLER,
  SAMPLED_IMAGE, STORAGE_IMAGE,
  UNIFORM_TEXEL_BUFFER,
  STORAGE_TEXEL_BUFFER,
  UNIFORM_BUFFER[_DYNAMIC],
  STORAGE_BUFFER[_DYNAMIC],
  INPUT_ATTACHMENT
```

```
typedef struct VkDescriptorSetLayoutBinding {
    uint32_t binding;
    VkDescriptorType descriptorType;  P.12
    uint32_t descriptorCount;
    VkShaderStageFlags stageFlags;  P.15
    const VkSampler* pImmutableSamplers;
} VkDescriptorSetLayoutBinding;
```

```
typedef struct VkDescriptorSetLayoutCreateInfo {
    VkStructureType sType;  P.15
    const void* pNext;
    VkDescriptorSetLayoutCreateFlags flags;
    uint32_t bindingCount;
    const VkDescriptorSetLayoutBinding* pBindings;
} VkDescriptorSetLayoutCreateInfo;
```

```
typedef struct VkDeviceGroupBindSparseInfo {
    VkStructureType sType;  P.15
    const void* pNext;
    uint32_t resourceDeviceIndex;
    uint32_t memoryDeviceIndex;
} VkDeviceGroupBindSparseInfo;
```

```
typedef struct VkDeviceGroupCommandBufferBeginInfo {
    VkStructureType sType;  P.15
    const void* pNext;
    uint32_t deviceMask;
} VkDeviceGroupCommandBufferBeginInfo;
```

```
typedef struct VkDeviceGroupDeviceCreateInfo {
    VkStructureType sType;  P.15
    const void* pNext; uint32_t physicalDeviceCount;
    const VkPhysicalDevice* pPhysicalDevices;
} VkDeviceGroupDeviceCreateInfo;
```

enum **VkDeviceGroupPresentModeFlagBitsKHR**:

```
VK_DEVICE_GROUP_PRESENT_MODE_X_BIT_KHR where X is
  LOCAL, REMOTE, SUM, LOCAL_MULTI_DEVICE
```

```
typedef struct VkDeviceGroupRenderPassBeginInfo {
    VkStructureType sType;  P.15
    const void* pNext; uint32_t deviceMask;
    uint32_t deviceRenderAreaCount;
    const VkRect2D* pDeviceRenderAreas;  P.15
} VkDeviceGroupRenderPassBeginInfo;
```

```
typedef struct VkDeviceGroupSubmitInfo {
    VkStructureType sType;  P.15
    const void* pNext;
    uint32_t waitSemaphoreCount;
    const uint32_t* pWaitSemaphoreDeviceIndices;
    uint32_t commandBufferCount;
    const uint32_t* pCommandBufferDeviceMasks;
    uint32_t signalSemaphoreCount;
    const uint32_t* pSignalSemaphoreDeviceIndices;
} VkDeviceGroupSubmitInfo;
```

```
typedef struct VkExportFenceCreateInfo {
    VkStructureType sType;  P.15
    const void* pNext;
    VkExternalFenceHandleTypeFlags handleTypes;  P.12
} VkExportFenceCreateInfo;
```

```
typedef struct VkExportMemoryAllocateInfo {
    VkStructureType sType;  P.15
    const void* pNext;
    VkExternalMemoryHandleTypeFlags handleTypes;  P.12
} VkExportMemoryAllocateInfo;
```

```
typedef struct VkExportSemaphoreCreateInfo {
    VkStructureType sType;  P.15
    const void* pNext;
    VkExternalSemaphoreHandleTypeFlags handleTypes;  P.13
} VkExportSemaphoreCreateInfo;
```

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

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

enum **VkExternalFenceHandleTypeFlagBits**:

```
  VK_EXTERNAL_FENCE_HANDLE_TYPE_X_BIT where X is
  OPAQUE_FD,
  OPAQUE_WIN32,
  OPAQUE_WIN32_KMT,
  SYNC_FD
```

```
typedef struct VkExternalImageFormatProperties {
    VkStructureType sType;  P.15
    void* pNext;
    VkExternalMemoryProperties
            externalMemoryProperties;  P.13
} VkExternalImageFormatProperties;
```

```
typedef struct VkExternalMemoryBufferCreateInfo {
    VkStructureType sType;  P.15
    const void* pNext;
    VkExternalMemoryHandleTypeFlags handleTypes;  P.12
} VkExternalMemoryBufferCreateInfo;
```

enum **VkExternalMemoryFeatureFlagBits**:

```
  VK_EXTERNAL_MEMORY_FEATURE_X_BIT where X is
  DEDICATED_ONLY,
  EXPORTABLE,
  IMPORTABLE
```

enum **VkExternalMemoryHandleTypeFlagBits**:

```
  VK_EXTERNAL_MEMORY_HANDLE_TYPE_X_BIT where X is
  OPAQUE_FD,
  OPAQUE_WIN32,
  OPAQUE_WIN32_KMT,
  D3D11_TEXTURE,
  D3D11_TEXTURE_KMT,
  D3D12_HEAP,
  D3D12_RESOURCE
```

Continued on next page >

Structures and Enumerations (continued)

typedef struct **VkExternalMemoryImageCreateInfo** {
VkStructureType *sType*; P.15
const void* *pNext*;
VkExternalMemoryHandleTypeFlags *handleTypes*; P.12
} VkExternalMemoryImageCreateInfo;

typedef struct **VkExternalMemoryProperties** {
VkExternalMemoryFeatureFlags
externalMemoryFeatures; P.12
VkExternalMemoryHandleTypeFlags
exportFromImportedHandleTypes; P12
VkExternalMemoryHandleTypeFlags
compatibleHandleTypes; P12
} VkExternalMemoryProperties;

enum **VkExternalSemaphoreHandleTypeFlagBits**:
VK_EXTERNAL_SEMAPHORE_HANDLE_TYPE_*X*_BIT where *X* is OPAQUE_FD, OPAQUE_WIN32, OPAQUE_WIN32_KMT, D3D12_FENCE, SYNC_FD

enum **VkFormat**:
VK_FORMAT_*X* where *X* is
UNDEFINED,
R4G4_UNORM_PACK8,
R4G4B4A4_UNORM_PACK16,
B4G4R4A4_UNORM_PACK16,
R5G6B5_UNORM_PACK16,
B5G6R5_UNORM_PACK16,
R5G5B5A1_UNORM_PACK16,
B5G5R5A1_UNORM_PACK16,
A1R5G5B5_UNORM_PACK16,
R8_[UNORM, SNORM, USCALED],
R8_[SSCALED, UINT, SINT, SRGB],
R8G8_[UNORM, SNORM, USCALED],
R8G8_[SSCALED, UINT, SINT, SRGB],
R8G8B8_[UNORM, SNORM, USCALED],
R8G8B8_[SSCALED, UINT, SINT, SRGB],
B8G8R8_[UNORM, SNORM, USCALED],
B8G8R8_[SSCALED, UINT, SINT, SRGB],
R8G8B8A8_[UNORM, SNORM, USCALED],
R8G8B8A8_[SSCALED, UINT, SINT, SRGB],
B8G8R8A8_[UNORM, SNORM, USCALED],
B8G8R8A8_[SSCALED, UINT, SINT, SRGB],
A8B8G8R8_[UNORM, SNORM, USCALED]_PACK32,
A8B8G8R8_[SSCALED, UINT, SINT, SRGB]_PACK32,
A2R10G10B10_[UNORM, SNORM, USCALED]_PACK32,
A2R10G10B10_[SSCALED, UINT, SINT]_PACK32,
A2B10G10R10_[UNORM, SNORM, USCALED]_PACK32,
A2B10G10R10_[SSCALED, UINT, SINT]_PACK32,
R16_[UNORM, SNORM, USCALED],
R16_[SSCALED, UINT, SINT, SFLOAT],
R16G16_[UNORM, SNORM, USCALED],
R16G16_[SSCALED, UINT, SINT, SFLOAT],
R16G16B16_[UNORM, SNORM, USCALED],
R16G16B16_[SSCALED, UINT, SINT, SFLOAT],
R16G16B16A16_[UNORM, SNORM, USCALED],
R16G16B16A16_[SSCALED, UINT, SINT, SFLOAT],
R32_[UINT, SINT, SFLOAT],
R32G32_[UINT, SINT, SFLOAT],
R32G32B32_[UINT, SINT, SFLOAT],
R32G32B32A32_[UINT, SINT, SFLOAT],
R64_[UINT, SINT, SFLOAT],
R64G64_[UINT, SINT, SFLOAT],
R64G64B64_[UINT, SINT, SFLOAT],
R64G64B64A64_[UINT, SINT, SFLOAT],
B10G11R11_UFLOAT_PACK32,
E5B9G9R9_UFLOAT_PACK32,
X8_D24_UNORM_PACK32,
D32_SFLOAT[_S8_UINT],
S8_UINT,
D[16, 24]_UNORM_S8_UINT,
BC1_[RGB, RGBA]_UNORM_BLOCK,
BC1_[RGB, RGBA]_SRGB_BLOCK,
BC2_[UNORM, SRGB]_BLOCK,
BC3_[UNORM, SRGB]_BLOCK,
BC4_[UNORM, SRGB]_BLOCK,
BC5_[UNORM, SRGB]_BLOCK,
BC6H_[UFLOAT, SFLOAT]_BLOCK,
BC7_[UNORM, SRGB]_BLOCK,
ETC2_R8G8B8_[UNORM, SRGB]_BLOCK,
ETC2_R8G8B8A1_[UNORM, SRGB]_BLOCK,
ETC2_R8G8B8A8_[UNORM, SRGB]_BLOCK,
EAC_R11_[UNORM, SRGB]_BLOCK,
EAC_R11G11_[UNORM, SRGB]_BLOCK,
ASTC_4x4_[UNORM, SRGB]_BLOCK,
ASTC_5x4_[UNORM, SRGB]_BLOCK,
ASTC_5x5_[UNORM, SRGB]_BLOCK,
ASTC_6x5_[UNORM, SRGB]_BLOCK,
ASTC_6x6_[UNORM, SRGB]_BLOCK,
ASTC_8x5_[UNORM, SRGB]_BLOCK,
ASTC_8x6_[UNORM, SRGB]_BLOCK,
ASTC_8x8_[UNORM, SRGB]_BLOCK,
ASTC_10x5_[UNORM, SRGB]_BLOCK,
ASTC_10x6_[UNORM, SRGB]_BLOCK,
ASTC_10x8_[UNORM, SRGB]_BLOCK,
ASTC_10x10_[UNORM, SRGB]_BLOCK,
ASTC_12x10_[UNORM, SRGB]_BLOCK,
ASTC_12x12_[UNORM, SRGB]_BLOCK,
G8B8G8R8_422_UNORM,
B8G8R8G8_422_UNORM,
G8_B8_R8_3PLANE_420_UNORM,
G8_B8R8_2PLANE_{420, 422}_UNORM,
G8_B8_R8_3PLANE_{422, 444}_UNORM,
R10X6_UNORM_PACK16,
R10X6G10X6_UNORM_2PACK16,
R10X6G10X6B10X6A10X6_UNORM_4PACK16,
G10X6B10X6G10X6R10X6_422_UNORM_4PACK16,
B10X6G10X6R10X6G10X6_422_UNORM_4PACK16,
G10X6_B10X6_R10X6_3PLANE_420_UNORM_3PACK16,
G10X6_B10X6R10X6_2PLANE_420_UNORM_3PACK16,
G10X6_B10X6_R10X6_3PLANE_422_UNORM_3PACK16,
G10X6_B10X6R10X6_2PLANE_422_UNORM_3PACK16,
G10X6_B10X6_R10X6_3PLANE_444_UNORM_3PACK16,
R12X4_UNORM_PACK16,
R12X4G12X4_UNORM_2PACK16,
R12X4G12X4B12X4A12X4_UNORM_4PACK16,
G12X4B12X4G12X4R12X4_422_UNORM_4PACK16,
B12X4G12X4R12X4G12X4_422_UNORM_4PACK16,
G12X4_B12X4_R12X4_3PLANE_420_UNORM_3PACK16,
G12X4_B12X4R12X4_2PLANE_{420, 422}_UNORM_3PACK16,
G12X4_B12X4_R12X4_3PLANE_422_UNORM_3PACK16,
G12X4_B12X4_R12X4_3PLANE_444_UNORM_3PACK16,
G16B16G16R16_422_UNORM,
B16G16R16G16_422_UNORM,
G16_B16_R16_3PLANE_{420, 422, 444}_UNORM,
G16_B16R16_2PLANE_{420, 422}_UNORM

enum **VkImageAspectFlagBits**:
VK_IMAGE_ASPECT_*X*_BIT where *X* is
COLOR, DEPTH, STENCIL, METADATA, PLANE_[0,1,2]

enum **VkImageCreateFlagBits**:
VK_IMAGE_CREATE_*X*_BIT where *X* is
SPARSE_{BINDING, RESIDENCY, ALIASED},
MUTABLE_FORMAT,
{CUBE, 2D_ARRAY, TEXEL_VIEW}_COMPATIBLE,
ALIAS, BIND_SFR,
EXTENDED_USAGE,
PROTECTED,
DISJOINT

enum **VkImageLayout**:
VK_IMAGE_LAYOUT_*X* where *X* is
UNDEFINED, GENERAL, PREINITIALIZED,
COLOR_ATTACHMENT_OPTIMAL,
DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL,
DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL,
DEPTH_STENCIL_ATTACHMENT_OPTIMAL,
DEPTH_STENCIL_READ_ONLY_OPTIMAL,
SHADER_READ_ONLY_OPTIMAL,
TRANSFER_{SRC, DST}_OPTIMAL,
DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL,
DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL,
VK_IMAGE_LAYOUT_PRESENT_SRC_KHR,
VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR

NOTE: For the functions vkCmdCopyImage, vkCmdCopyBufferToImage, vkCmdCopyImageToBuffer, vkCmdBlitImage, and vkCmdResolveImage, the enum VkImageLayout for the following parameters may be:

srcImageLayout: VK_IMAGE_LAYOUT_GENERAL,
VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL,
VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR

dstImageLayout: VK_IMAGE_LAYOUT_GENERAL,
VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR

typedef struct **VkImageMemoryBarrier** {
VkStructureType *sType*; P.15
const void* *pNext*;
VkAccessFlags *srcAccessMask*; P.12
VkAccessFlags *dstAccessMask*; P.12
VkImageLayout *oldLayout*; P.13
VkImageLayout *newLayout*; P.13
uint32_t *srcQueueFamilyIndex*;
uint32_t *dstQueueFamilyIndex*;
VkImage *image*;
VkImageSubresourceRange *subresourceRange*; P.1313
} VkImageMemoryBarrier;

typedef struct **VkImagePlaneMemoryRequirementsInfo** {
VkStructureType *sType*; P.15
const void* *pNext*;
VkImageAspectFlagBits *planeAspect*; P.1313
} VkImagePlaneMemoryRequirementsInfo;

typedef struct **VkImageSubresourceLayers** {
VkImageAspectFlags *aspectMask*; P.13
uint32_t *mipLevel*;
uint32_t *baseArrayLayer*;
uint32_t *layerCount*;
} VkImageSubresourceLayers;

typedef struct **VkImageSubresourceRange** {
VkImageAspectFlags *aspectMask*; P.13
uint32_t *baseMipLevel*;
uint32_t *levelCount*;
uint32_t *baseArrayLayer*;
uint32_t *layerCount*;
} VkImageSubresourceRange;

enum **VkImageTiling**:
VK_IMAGE_TILING_{OPTIMAL, LINEAR}

enum **VkImageType**:
VK_IMAGE_TYPE_{1D, 2D, 3D}

enum **VkImageUsageFlagBits**:
VK_IMAGE_USAGE_*X*_BIT where *X* is
TRANSFER_SRC,
TRANSFER_DST,
SAMPLED,
STORAGE,
COLOR_ATTACHMENT,
DEPTH_STENCIL_ATTACHMENT,
TRANSIENT_ATTACHMENT,
INPUT_ATTACHMENT

typedef struct **VkImageViewUsageCreateInfo** {
VkStructureType *sType*; P.15
const void* *pNext*;
VkImageUsageFlags *usage*; P.13
} VkImageViewUsageCreateInfo;

typedef struct **VkInputAttachmentAspectReference** {
uint32_t *subpass*;
uint32_t *inputAttachmentIndex*;
VkImageAspectFlags *aspectMask*; P.13
} VkInputAttachmentAspectReference;

typedef struct **VkMemoryAllocateFlagsInfo** {
VkStructureType *sType*; P.15
const void* *pNext*;
VkMemoryAllocateFlags *flags*;
uint32_t *deviceMask*;
} VkMemoryAllocateFlagsInfo;
flags: VK_MEMORY_ALLOCATE_DEVICE_MASK_BIT

typedef struct **VkMemoryBarrier** {
VkStructureType *sType*; P.15
const void* *pNext*;
VkAccessFlags *srcAccessMask*; P.12
VkAccessFlags *dstAccessMask*; P.12
} VkMemoryBarrier;

typedef struct **VkMemoryDedicatedAllocateInfo** {
VkStructureType *sType*; P.15
const void* pNext;
VkImage *image*; VkBuffer *buffer*;
} VkMemoryDedicatedAllocateInfo;

typedef struct **VkMemoryDedicatedRequirements** {
VkStructureType *sType*; P.15
void* *pNext*;
VkBool32 *prefersDedicatedAllocation*;
VkBool32 *requiresDedicatedAllocation*;
} VkMemoryDedicatedRequirements;

typedef struct **VkMemoryHeap** {
VkDeviceSize *size*;
VkMemoryHeapFlags *flags*;
} VkMemoryHeap;
flags: VK_MEMORY_HEAP_*X*_BIT where *X* is
DEVICE_LOCAL, MULTI_INSTANCE

typedef struct **VkMemoryRequirements** {
VkDeviceSize *size*;
VkDeviceSize *alignment*;
uint32_t *memoryTypeBits*;
} VkMemoryRequirements;

typedef struct **VkMemoryRequirements2** {
VkStructureType *sType*; P.15
void* *pNext*;
VkMemoryRequirements *memoryRequirements*; P.13
} VkMemoryRequirements2;
pNext must be NULL or point to:
VkMemoryDedicatedRequirements P13

Continued on next page >

Structures and Enumerations (continued)

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

propertyFlags: VK_MEMORY_PROPERTY_*X*_BIT where *X* is DEVICE_LOCAL, HOST_VISIBLE, HOST_COHERENT, HOST_CACHED, LAZILY_ALLOCATED, PROTECTED

```
typedef struct VkOffset2D {
    int32_t x;
    int32_t y;
} VkOffset2D;

typedef struct VkOffset3D {
    int32_t x;
    int32_t y;
    int32_t z;
} VkOffset3D;

typedef struct VkPhysicalDevice16BitStorageFeatures {
    VkStructureType sType; P.15
    void* pNext;
    VkBool32 storageBuffer16BitAccess;
    VkBool32 uniformAndStorageBuffer16BitAccess;
    VkBool32 storagePushConstant16;
    VkBool32 storageInputOutput16;
} VkPhysicalDevice16BitStorageFeatures;

typedef struct VkPhysicalDeviceExternalImageFormatInfo {
    VkStructureType sType; P.15
    const void* pNext;
    VkExternalMemoryHandleTypeFlagBits handleType; P.1512
} VkPhysicalDeviceExternalImageFormatInfo;

typedef struct VkPhysicalDeviceFeatures {
    VkBool32 robustBufferAccess;
    VkBool32 fullDrawIndexUint32;
    VkBool32 imageCubeArray;
    VkBool32 independentBlend;
    VkBool32 geometryShader;
    VkBool32 tessellationShader;
    VkBool32 sampleRateShading;
    VkBool32 dualSrcBlend;
    VkBool32 logicOp;
    VkBool32 multiDrawIndirect;
    VkBool32 drawIndirectFirstInstance;
    VkBool32 depthClamp;
    VkBool32 depthBiasClamp;
    VkBool32 fillModeNonSolid;
    VkBool32 depthBounds;
    VkBool32 wideLines;
    VkBool32 largePoints;
    VkBool32 alphaToOne;
    VkBool32 multiViewport;
    VkBool32 samplerAnisotropy;
    VkBool32 textureCompressionETC2;
    VkBool32 textureCompressionASTC_LDR;
    VkBool32 textureCompressionBC;
    VkBool32 occlusionQueryPrecise;
    VkBool32 pipelineStatisticsQuery;
    VkBool32 vertexPipelineStoresAndAtomics;
    VkBool32 fragmentStoresAndAtomics;
    VkBool32 shaderTessellationAndGeometryPointSize;
    VkBool32 shaderImageGatherExtended;
    VkBool32 shaderStorageImageExtendedFormats;
    VkBool32 shaderStorageImageMultisample;
    VkBool32 shaderStorageImageReadWithoutFormat;
    VkBool32 shaderStorageImageWriteWithoutFormat;
    VkBool32 shaderUniformBufferArrayDynamicIndexing;
    VkBool32 shaderSampledImageArrayDynamicIndexing;
    VkBool32 shaderStorageBufferArrayDynamicIndexing;
    VkBool32 shaderStorageImageArrayDynamicIndexing;
    VkBool32 shaderClipDistance;
    VkBool32 shaderCullDistance;
    VkBool32 shaderFloat64;
    VkBool32 shaderInt64;
    VkBool32 shaderInt16;
    VkBool32 shaderResourceResidency;
    VkBool32 shaderResourceMinLod;
    VkBool32 sparseBinding;
    VkBool32 sparseResidencyBuffer;
    VkBool32 sparseResidencyImage2D;
    VkBool32 sparseResidencyImage3D;
    VkBool32 sparseResidency2Samples;
    VkBool32 sparseResidency4Samples;
    VkBool32 sparseResidency8Samples;
    VkBool32 sparseResidency16Samples;
    VkBool32 sparseResidencyAliased;
    VkBool32 variableMultisampleRate;
    VkBool32 inheritedQueries;
} VkPhysicalDeviceFeatures;

typedef struct VkPhysicalDeviceFeatures2 {
    VkStructureType sType; P.15
    void* pNext;
    VkPhysicalDeviceFeatures features; P.14
} VkPhysicalDeviceFeatures2;
```

pNext must be NULL or point to one of:
VkPhysicalDevice16BitStorageFeatures P.14
VkPhysicalDeviceMultiviewFeatures P.14
VkPhysicalDeviceProtectedMemoryFeatures P.15
VkPhysicalDeviceSamplerYcbcrConversionFeatures P.15
VkPhysicalDeviceShaderDrawParameterFeatures P.15
VkPhysicalDeviceVariablePointerFeatures P.15

```
typedef struct VkPhysicalDeviceIDProperties {
    VkStructureType sType; P.15
    void* pNext;
    uint8_t deviceUUID[VK_UUID_SIZE];
    uint8_t driverUUID[VK_UUID_SIZE];
    uint8_t deviceLUID[VK_LUID_SIZE];
    uint32_t deviceNodeMask;
    VkBool32 deviceLUIDValid;
} VkPhysicalDeviceIDProperties;

typedef struct VkPhysicalDeviceLimits {
    uint32_t maxImageDimension1D;
    uint32_t maxImageDimension2D;
    uint32_t maxImageDimension3D;
    uint32_t maxImageDimensionCube;
    uint32_t maxImageArrayLayers;
    uint32_t maxTexelBufferElements;
    uint32_t maxUniformBufferRange;
    uint32_t maxStorageBufferRange;
    uint32_t maxPushConstantsSize;
    uint32_t maxMemoryAllocationCount;
    uint32_t maxSamplerAllocationCount;
    VkDeviceSize bufferImageGranularity;
    VkDeviceSize sparseAddressSpaceSize;
    uint32_t maxBoundDescriptorSets;
    uint32_t maxPerStageDescriptorSamplers;
    uint32_t maxPerStageDescriptorUniformBuffers;
    uint32_t maxPerStageDescriptorStorageBuffers;
    uint32_t maxPerStageDescriptorSampledImages;
    uint32_t maxPerStageDescriptorStorageImages;
    uint32_t maxPerStageDescriptorInputAttachments;
    uint32_t maxPerStageResources;
    uint32_t maxDescriptorSetSamplers;
    uint32_t maxDescriptorSetUniformBuffers;
    uint32_t maxDescriptorSetUniformBuffersDynamic;
    uint32_t maxDescriptorSetStorageBuffers;
    uint32_t maxDescriptorSetStorageBuffersDynamic;
    uint32_t maxDescriptorSetSampledImages;
    uint32_t maxDescriptorSetStorageImages;
    uint32_t maxDescriptorSetInputAttachments;
    uint32_t maxVertexInputAttributes;
    uint32_t maxVertexInputBindings;
    uint32_t maxVertexInputAttributeOffset;
    uint32_t maxVertexInputBindingStride;
    uint32_t maxVertexOutputComponents;
    uint32_t maxTessellationGenerationLevel;
    uint32_t maxTessellationPatchSize;
    uint32_t
        maxTessellationControlPerVertexInputComponents;
    uint32_t
      maxTessellationControlPerVertexOutputComponents;
    uint32_t
       maxTessellationControlPerPatchOutputComponents;
    uint32_t maxTessellationControlTotalOutputComponents;
    uint32_t maxTessellationEvaluationInputComponents;
    uint32_t maxTessellationEvaluationOutputComponents;
    uint32_t maxGeometryShaderInvocations;
    uint32_t maxGeometryInputComponents;
    uint32_t maxGeometryOutputComponents;
    uint32_t maxGeometryOutputVertices;
    uint32_t maxGeometryTotalOutputComponents;
    uint32_t maxFragmentInputComponents;
    uint32_t maxFragmentOutputAttachments;
    uint32_t maxFragmentDualSrcAttachments;
    uint32_t maxFragmentCombinedOutputResources;
    uint32_t maxComputeSharedMemorySize;
    uint32_t maxComputeWorkGroupCount[3];
    uint32_t maxComputeWorkGroupInvocations;
    uint32_t maxComputeWorkGroupSize[3];
    uint32_t subPixelPrecisionBits;
    uint32_t subTexelPrecisionBits;
    uint32_t mipmapPrecisionBits;
    uint32_t maxDrawIndexedIndexValue;
    uint32_t maxDrawIndirectCount;
    float maxSamplerLodBias;
    float maxSamplerAnisotropy;
    uint32_t maxViewports;
    uint32_t maxViewportDimensions[2];
    float viewportBoundsRange[2];
    uint32_t viewportSubPixelBits;
    size_t minMemoryMapAlignment;
    VkDeviceSize minTexelBufferOffsetAlignment;
    VkDeviceSize minUniformBufferOffsetAlignment;
    VkDeviceSize minStorageBufferOffsetAlignment;
    int32_t minTexelOffset; uint32_t maxTexelOffset;
    int32_t minTexelGatherOffset;
    uint32_t maxTexelGatherOffset;
    float minInterpolationOffset;
    float maxInterpolationOffset;
    uint32_t subPixelInterpolationOffsetBits;
    uint32_t maxFramebufferWidth;
    uint32_t maxFramebufferHeight;
    uint32_t maxFramebufferLayers;
    VkSampleCountFlags framebufferColorSampleCounts; P.15
    VkSampleCountFlags framebufferDepthSampleCounts; P.15
    VkSampleCountFlags framebufferStencilSampleCounts; P.15
    VkSampleCountFlags
        framebufferNoAttachmentsSampleCounts; P.15
    uint32_t maxColorAttachments;
    VkSampleCountFlags
        sampledImageColorSampleCounts; P.15
    VkSampleCountFlags
        sampledImageIntegerSampleCounts; P.15
    VkSampleCountFlags
        sampledImageDepthSampleCounts; P.15
    VkSampleCountFlags
        sampledImageStencilSampleCounts; P.15
    VkSampleCountFlags storageImageSampleCounts; P.15
    uint32_t maxSampleMaskWords;
    VkBool32 timestampComputeAndGraphics;
    float timestampPeriod;
    uint32_t maxClipDistances;
    uint32_t maxCullDistances;
    uint32_t maxCombinedClipAndCullDistances;
    uint32_t discreteQueuePriorities;
    float pointSizeRange[2]; float lineWidthRange[2];
    float pointSizeGranularity;
    float lineWidthGranularity;
    VkBool32 strictLines;
    VkBool32 standardSampleLocations;
    VkDeviceSize optimalBufferCopyOffsetAlignment;
    VkDeviceSize optimalBufferCopyRowPitchAlignment;
    VkDeviceSize nonCoherentAtomSize;
} VkPhysicalDeviceLimits;

typedef struct VkPhysicalDeviceMaintenance3Properties {
    VkStructureType sType; P.15
    void* pNext;
    uint32_t maxPerSetDescriptors;
    VkDeviceSize maxMemoryAllocationSize;
} VkPhysicalDeviceMaintenance3Properties;

typedef struct VkPhysicalDeviceMemoryProperties {
    uint32_t memoryTypeCount; VkMemoryType
        memoryTypes[VK_MAX_MEMORY_TYPES]; P.14
    uint32_t memoryHeapCount; VkMemoryHeap
        memoryHeaps[VK_MAX_MEMORY_HEAPS]; P.13
} VkPhysicalDeviceMemoryProperties;

typedef struct VkPhysicalDeviceMultiviewFeatures {
    VkStructureType sType; P.15
    void* pNext;
    VkBool32 multiview;
    VkBool32 multiviewGeometryShader;
    VkBool32 multiviewTessellationShader;
} VkPhysicalDeviceMultiviewFeatures;

typedef struct VkPhysicalDeviceMultiviewProperties {
    VkStructureType sType; P.15
    void* pNext;
    uint32_t maxMultiviewViewCount;
    uint32_t maxMultiviewInstanceIndex;
} VkPhysicalDeviceMultiviewProperties;

typedef struct VkPhysicalDevicePointClippingProperties {
    VkStructureType sType; P.15
    void* pNext;
    VkPointClippingBehavior pointClippingBehavior;
} VkPhysicalDevicePointClippingProperties;
```

pointClippingBehavior: VK_POINT_CLIPPING_BEHAVIOR_X where X is ALL_CLIP_PLANES, USER_CLIP_PLANES_ONLY

```
typedef struct VkPhysicalDeviceProperties {
    uint32_t apiVersion; uint32_t driverVersion;
    uint32_t vendorID; uint32_t deviceID;
    VkPhysicalDeviceType deviceType;
    char deviceName[
        VK_MAX_PHYSICAL_DEVICE_NAME_SIZE];
    uint8_t pipelineCacheUUID[VK_UUID_SIZE];
    VkPhysicalDeviceLimits limits; P.14
    VkPhysicalDeviceSparseProperties sparseProperties; P.15
} VkPhysicalDeviceProperties;
```

deviceType: VK_PHYSICAL_DEVICE_TYPE_*X* where *X* is OTHER, INTEGRATED_GPU, DISCRETE_GPU, VIRTUAL_GPU, CPU

Continued on next page >

Structures and Enumerations (continued)

```
typedef struct
    VkPhysicalDeviceProtectedMemoryFeatures {
        VkStructureType sType;  P.15
        void* pNext;  VkBool32 protectedMemory;
    } VkPhysicalDeviceProtectedMemoryFeatures;
```

```
typedef struct
    VkPhysicalDeviceProtectedMemoryProperties {
        VkStructureType sType;  P.15
        void* pNext; VkBool32 protectedNoFault;
    } VkPhysicalDeviceProtectedMemoryProperties;
```

```
typedef struct
    VkPhysicalDeviceSamplerYcbcrConversionFeatures {
        VkStructureType sType;  P.15
        void* pNext;
        VkBool32 samplerYcbcrConversion;
    } VkPhysicalDeviceSamplerYcbcrConversionFeatures;
```

```
typedef struct
    VkPhysicalDeviceShaderDrawParameterFeatures {
        VkStructureType sType;  P.15
        void* pNext;
        VkBool32 shaderDrawParameters;
    } VkPhysicalDeviceShaderDrawParameterFeatures;
```

```
typedef struct VkPhysicalDeviceSparseProperties {
    VkBool32 residencyStandard2DBlockShape;
    VkBool32 residencyStandard2DMultisampleBlockShape;
    VkBool32 residencyStandard3DBlockShape;
    VkBool32 residencyAlignedMipSize;
    VkBool32 residencyNonResidentStrict;
} VkPhysicalDeviceSparseProperties;
```

```
typedef struct VkPhysicalDeviceSubgroupProperties {
    VkStructureType sType;  P.15
    void* pNext; uint32_t subgroupSize;
    VkShaderStageFlags supportedStages;  P.15
    VkSubgroupFeatureFlags supportedOperations;
    VkBool32 quadOperationsInAllStages;
} VkPhysicalDeviceSubgroupProperties;
```

supportedOperations:
VK_SUBGROUP_FEATURE_*X*_BIT where *X* is
ARITHMETIC, BALLOT, BASIC, CLUSTERED, QUAD, SHUFFLE,
SHUFFLE_RELATIVE, VOTE

```
typedef struct VkPhysicalDeviceSurfaceInfo2KHR {
    VkStructureType sType;  P.15
    const void* pNext; VkSurfaceKHR surface;
} VkPhysicalDeviceSurfaceInfo2KHR;
```

```
typedef struct VkPhysicalDeviceVariablePointerFeatures {
    VkStructureType sType;  P.15
    void* pNext;
    VkBool32 variablePointersStorageBuffer;
    VkBool32 variablePointers;
} VkPhysicalDeviceVariablePointerFeatures;
```

enum **VkPipelineBindPoint**:
VK_PIPELINE_BIND_POINT_COMPUTE,
VK_PIPELINE_BIND_POINT_GRAPHICS

enum **VkPipelineCreateFlagBits**:
VK_PIPELINE_CREATE_*X* where *X* is
DISABLE_OPTIMIZATION_BIT,
ALLOW_DERIVATIVES_BIT,
DERIVATIVE_BIT,
VIEW_INDEX_FROM_DEVICE_INDEX_BIT,
DISPATCH_BASE

```
typedef struct VkPipelineShaderStageCreateInfo {
    VkStructureType sType;  P.15
    const void* pNext;
    VkPipelineShaderStageCreateFlags flags;  = 0
    VkShaderStageFlagBits stage;  P.15
    VkShaderModule module;
    const char* pName;
    const VkSpecializationInfo* pSpecializationInfo;  P.15
} VkPipelineShaderStageCreateInfo;
```

```
typedef struct VkSpecializationInfo {
    uint32_t mapEntryCount;
    const VkSpecializationMapEntry* pMapEntries;  P.15
    size_t dataSize; const void* pData;
} VkSpecializationInfo;
```

```
typedef struct VkSpecializationMapEntry {
    uint32_t constantID;
    uint32_t offset;
    size_t size;
} VkSpecializationMapEntry;
```

enum **VkPipelineStageFlagBits**:
VK_PIPELINE_STAGE_*X*_BIT where *X* is
TOP_OF_PIPE, DRAW_INDIRECT, VERTEX_[INPUT, SHADER],
TESSELLATION_[CONTROL, EVALUATION]_SHADER,
[COMPUTE, GEOMETRY, FRAGMENT]_SHADER,
[EARLY, LATE]_FRAGMENT_TESTS,
COLOR_ATTACHMENT_OUTPUT,
TRANSFER, BOTTOM_OF_PIPE, HOST,
ALL_{GRAPHICS, COMMANDS}

```
typedef struct
    VkPipelineTessellationDomainOriginStateCreateInfo{
        VkStructureType sType;  P.15
        const void* pNext;
        VkTessellationDomainOrigin domainOrigin;
    } VkPipelineTessellationDomainOriginStateCreateInfo;
```

domainOrigin:
VK_TESSELLATION_DOMAIN_ORIGIN_UPPER_LEFT
VK_TESSELLATION_DOMAIN_ORIGIN_LOWER_LEFT

```
typedef struct VkProtectedSubmitInfo {
    VkStructureType sType;  P.15
    const void* pNext;  VkBool32 protectedSubmit;
} VkProtectedSubmitInfo;
```

enum **VkQueryPipelineStatisticFlagBits**:
VK_QUERY_PIPELINE_STATISTIC_*X*_BIT where *X* is
INPUT_ASSEMBLY_{VERTICES, PRIMITIVES},
VERTEX_SHADER_INVOCATIONS,
GEOMETRY_SHADER_{INVOCATIONS, PRIMITIVES},
CLIPPING_{INVOCATIONS, PRIMITIVES},
FRAGMENT_SHADER_INVOCATIONS,
TESSELLATION_CONTROL_SHADER_PATCHES,
TESSELLATION_EVALUATION_SHADER_INVOCATIONS,
COMPUTE_SHADER_INVOCATIONS

```
typedef struct VkRect2D {
    VkOffset2D offset;  P14
    VkExtent2D extent;  P.12
} VkRect2D;
```

```
typedef struct
    VkRenderPassInputAttachmentAspectCreateInfo {
        VkStructureType sType;  P.15
        const void* pNext;  uint32_t aspectReferenceCount;
        const VkInputAttachmentAspectReference*
            pAspectReferences;  P.13
    } VkRenderPassInputAttachmentAspectCreateInfo;
```

```
typedef struct VkRenderPassMultiviewCreateInfo {
    VkStructureType sType;  P.15
    const void* pNext; uint32_t subpassCount;
    const uint32_t* pViewMasks;
    uint32_t dependencyCount;
    const int32_t* pViewOffsets;
    uint32_t correlationMaskCount;
    const uint32_t* pCorrelationMasks;
} VkRenderPassMultiviewCreateInfo;
```

enum **VkSampleCountFlagBits**:
VK_SAMPLE_COUNT_*X*_BIT where *X* is
1, 2, 4, 8, 16, 32, 64

```
typedef struct
    VkSamplerYcbcrConversionImageFormatProperties {
        VkStructureType sType;  P.15
        void* pNext;
        uint32_t combinedImageSamplerDescriptorCount;
    } VkSamplerYcbcrConversionImageFormatProperties;
```

```
typedef struct VkSamplerYcbcrConversionInfo {
    VkStructureType sType;  P.15
    const void* pNext;
    VkSamplerYcbcrConversion conversion;
} VkSamplerYcbcrConversionInfo
```

enum **VkShaderStageFlagBits**:
VK_SHADER_STAGE_*X* where *X* is
{VERTEX, GEOMETRY, FRAGMENT, COMPUTE}_BIT,
TESSELLATION_CONTROL_BIT,
TESSELLATION_EVALUATION_BIT,
ALL_GRAPHICS, ALL

enum **VkSharingMode**:
VK_SHARING_MODE_EXCLUSIVE,
VK_SHARING_MODE_CONCURRENT

```
typedef struct VkSparseMemoryBind {
    VkDeviceSize resourceOffset;VkDeviceSize size;
    VkDeviceMemory memory; VkDeviceSize memoryOffset;
    VkSparseMemoryBindFlags flags;
} VkSparseMemoryBind;
```

flags: VK_SPARSE_MEMORY_BIND_METADATA_BIT

VkStructureType
The name of each **VkStructureType** value is obtained by taking the name of the structure, stripping the leading **Vk**, prefixing each capital letter with _, converting the entire resulting string to upper case, and prefixing it with **VK_STRUCTURE_TYPE_**.

```
typedef struct VkSurfaceCapabilitiesKHR {
    uint32_t minImageCount;
    uint32_t maxImageCount;
    VkExtent2D currentExtent;  P.12
    VkExtent2D minImageExtent;  P.12
    VkExtent2D maxImageExtent;  P.12
    uint32_t maxImageArrayLayers;
    VkSurfaceTransformFlagsKHR supportedTransforms;  P.15
    VkSurfaceTransformFlagBitsKHR currentTransform;  P.15
    VkCompositeAlphaFlagsKHR
        supportedCompositeAlpha;  P.12
    VkImageUsageFlags supportedUsageFlags;  P.13
} VkSurfaceCapabilitiesKHR;
```

enum **VkCompositeAlphaFlagBitsKHR**:
VK_COMPOSITE_ALPHA_*X*_BIT_KHR where *X* is
OPAQUE, PRE_MULTIPLIED, POST_MULTIPLIED, INHERIT

```
typedef struct VkSurfaceFormatKHR {
    VkFormat format;  P.13
    VkColorSpaceKHR colorSpace;
} VkSurfaceFormatKHR;
```

colorSpace: VK_COLOR_SPACE_SRGB_NONLINEAR_KHR

enum **VkSurfaceTransformFlagBitsKHR** {
VK_SURFACE_TRANSFORM_*X*_BIT_KHR where *X* is
IDENTITY,
ROTATE_{90, 180, 270},
HORIZONTAL_MIRROR
HORIZONTAL_MIRROR_ROTATE_{90, 180, 270},
INHERIT

```
typedef struct VkViewport {
    float x; float y;
    float width; float height;
    float minDepth; float maxDepth;
} VkViewport;
```

Learn more about Vulkan

Vulkan is maintained by the Khronos Group, a worldwide consortium of organizations that work to create and maintain key standards used across many industries. Visit Khronos online for resources to help you use and master Vulkan:

Main Vulkan Resource Page: khronos.org/vulkan/

Vulkan Registry: khronos.org/registry/vulkan/

Forums: forums.khronos.org/

Courses: khronos.org/developers/training/

Videos & Presentations: khr.io/library/

Khronos Events: khronos.org/news/events/

Khronos Blog: khronos.org/blog/

Reference Guides: khr.io/refguides/

Khronos Books: khronos.org/developers/books/

Khronos Merchandise: khronos.org/store/

@thekhronosgroup   khronos.org

Vulkan Reference Guide Index

The following index shows each item included on this card along with the page on which it is described. The color of the row in the table below is the color of the pane to which you should refer.

The Khronos Group is an industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics and dynamic media on a wide variety of platforms and devices.

See www.khronos.org to learn more about the Khronos Group.
See www.khronos.org/vulkan to learn more about Vulkan.

Reference guide production by Miller & Mattson www.millermattson.com