



Chapter 3: Common Patterns: *if someone missed to catch it yet* 😊

```
Object  Vk      VkInstance
Types   Vk      VkInstanceCreateInfo
Funcs   vk      vkCreateInstance()
Enums   VK_     VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO
```

Extensions

```
KHR:- Khronos authored,
EXT:- multi-company authored
```

Creating "VkZZZ" object

1. take `VkZZZCreateInfo` --> fill it up
2. call `vkCreateZZZ()`
3. also `vkDestroyZZZ()` before closing your app
4. Some objects get "allocated" rather than "created"


```
VkZZZAllocateInfo --> vkAllocateZZZ --> vkFreeZZZ
```
5. Sometimes there will be `.zzzCreateInfoCount` & `.pZZZCreateInfos`

```
e.g. .queueCreateInfoCount & .pQueueCreateInfos
```

 - > So you could like pass in an array/vector
 - > You will see this in lots of other places

Getting List/Properties

1. `vkEnumerateZZZ()` --> \see `[Chapter2.1.] vkEnumeratePhysicalDevices()` example

-- | -- | -- | -----

7. `sType` & `pNext`

- Many Vulkan structures include these two common fields

8. `sType` :-

- It may seem somewhat redundant, but this information can be useful for the `vulkan-loader` and actual `gpu-driver-implementations` to know what type of structure was passed in through `pNext`.

9. `pNext` :-

- allows to create a linked list between structures.
- It is mostly used when dealing with extensions that expose new structures to provide additional information to the `vulkan-loader`, `debugging-validation-layers`, and `gpu-driver-implementations`.
 - i.e. they can use the `pNext->sType` field to know what's ahead in the linked list

10. `pQueueCreateInfos` :- yes, you 'can' pass multiple 😊

- Sometimes there will be `.zzzCreateInfoCount` & `.pZZZCreateInfos`
 - So you could like pass in an array/vector
 - You will see this in lots of other places

-- | -- | -- | -----

11. CreateInfo StartingPoint

```
```cpp
VkRenderPassCreateInfo CI = {
 .sType = VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR,
 .pNext = nullptr,
 .flags = 0
};
```
```

12. Do remember to check the `'Valid Usage'` section within `each` manual-page



13. Getting/Enumerating VkObject list

```
uint32_t deviceCount = 0;
// [implicit valid usage]:- must be 0 [if 3rd-param = nullptr]
vkEnumeratePhysicalDevices(m_instance, &deviceCount, nullptr);
// it's kinda like the function is 'output-ing into' deviceCount



std::vector<VkPhysicalDevice> HardwareGPU_List(gpuCount);
// best to save this as a class member variable
vkEnumeratePhysicalDevices(m_instance, &deviceCount, HardwareGPU_List.data());
// note: it does return      VkResult return_code
```

-- | -- | -- | -----

14. Symbols :-

-  :- kinda means nothing
 - i kinda used to like make it look like a bit pattern-ish iguess 🤔🤔
-  :- "Yellow Card"
 - it means, you don't need to hesitate about this thingy right now 🤔 we will focus on this element later 🤔

```
1. ChapterZZZ => Unknown WIP/TBD Chapter
2. Chapter2.4 =>
  If LATER-CHAPTER => Dont hesitate right now, Do this when you reach that LATER-Chapter
  If PREV-CHAPTER => You can go back and check 🤔
    🔗 `SurfCAP.currentTransform`
    🔗 Chapter2.4
```

-  :- "Orange Card"
 - it means, this element is probably never gonna be 'necessary' for vulkan applications 🤔
-  ChapterZZZ
- 🔗 Chapter2.1
- 📄📄 Chapter2.1
 - `vkEnumeratePhysicalDevices()`
 - it means, Implement Exactly like in Chapter2.1 🤔
- 📄 REY_DOCS
 - Actual Notes
 - Mostly, vkdok.net documentation is good enough. But if I wanna add smth extra, it goes here
 - This section might get big & robust sometimes 🤔
- `</>` TheCode
- 📄 So far, The result
- 🤖 Visualization / [See it] / JSON Printing
- 📄🔗 2DriverIMPL
 - To The People Who are gonna Implement the Driver
 - Other Keyword:- "DriverGurantee"

-- | -- | -- | -----