Overview

We need to create/get hold of a couple of handles:		
Instance	1 `VkInstance` per program/app	`VkInstance`
Window Surface	*Surface(OS-Window)* [for actually linking Vulkan-Renders to Screen/Surface]	`VkSurfaceKHR`
Physical Device	*An Actual HARDWARE-GPU-device*	`VkPhysicalDevice`
Queue	*Queue(Commands)* to be executed on the GPU	`VkQueue`
Logical Device	*The "Logical" GPU Context/Interface* (Software Layer)	`VkDevice`
Swap Chain	*Sends Rendered-Image to the Surface(OS-Window)* / *Keeps a backup image-buffer to Render onto*	`VkSwapchainKHR`

Take a look into this awesome slide from slide-26 onwards, to understand what each of steps *"feel like"/mean/"how to imagine them"*.

*slide = Vulkanised 2023 Tutorial Part 1

Chapter 2: `VkDevice`

1. `vkEnumeratePhysicalDevices(m_instance, &m_deviceCount, nullptr)`
   - https://vkdoc.net/man/vkEnumeratePhysicalDevices
   - **REY_DOCs**

```
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
```

   - Visualization / [See it] / JSON Printing:- 4.guide.chapter2.1.json.hh
   - So far, The result:- 4.guide.chapter2.1.midway.hh

2. `vkCreateDevice()`

- https://vkdoc.net/man/vkCreateDevice
  - `param pAllocator` -> *"ChapterZZZ"*
- **REY_DOCs**
  - *we are not gonna call the* `vkCreateDevice()` *yeeeet....*
    - *but, yes, we've already made the class container around it* 😅
    - *we'll call this functiion in* `Chapter2.9.`
  - *but we did need to know first about* `vkCreateDevice()`
    - *because, the idea is, our sole task is to* ***fill it up step by step***

3. `VkDeviceCreateInfo`

- https://vkdoc.net/man/VkDeviceCreateInfo
  - `.LayerInfo` -> *Deprecated*
  - `.ExtensionInfo` -> *"ChapterZZZ"*
  - `.pQueueCreateInfos` -> *next part*
    - *So far, The result:-* *4.guide.chapter2.3.midway.hh*
- **REY_DOCs**
  - `.pQueueCreateInfos` -> *yes, you 'can' mass 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*

-- | -- | -- |

---

4. `VkDeviceQueueCreateInfo` - *'The Real Deal'*

- https://vkdoc.net/man/VkDeviceQueueCreateInfo
  - `.queueFamilyIndex` -> *next 3 subchapters*
    - *So far, The result:-* *4.guide.chapter2.4.midway.hh*
- **REY_DOCs:- Support for multiple QCI**
  - `.pQueuePriorities` -> *yes, this can be multiple "Priorities"* 😂 *[idk yet why tho]*

```
/* ============ REY_LoggerNUtils::REY_Utils.hh ============ */
REY_ArrayDYN<VkDeviceQueueCreateInfo> Array = REY_ArrayDYN<VkDeviceQueueCreateInfo>(2);
    // allocate enough space for 2 elements
REY_ARRAY_PUSH_BACK(Array) = this->Default_QCI;
REY_ARRAY_PUSH_BACK(Array) =          Your_QCI;

/* ============ std::vector ============ */
std::vector<VkDeviceQueueCreateInfo> Array = std::vector<VkDeviceQueueCreateInfo>(2);
Array.push_back(this->Default_QCI);
Array.push_back(         Your_QCI)
```

- So far, The result:- 4.guide.chapter2.4.TheEnd.hh

5. `vkGetPhysicalDeviceQueueFamilyProperties()`

- https://vkdoc.net/man/vkGetPhysicalDeviceQueueFamilyProperties
- **REY_DOCs**
  - *a GPU can have "multiple QueueFamilies"*
    - *a* `QueueFamily` *might support* `VK_QUEUE_GRAPHICS_BIT`

- *another* `QueueFamily` *might support* `VK_QUEUE_COMPUTE_BIT`
- *another* `QueueFamily` *might support* `VK_QUEUE_TRANSFER_BIT`
- *another* `QueueFamily` *might support* `VK_QUEUE_VIDEO_ENCODE_BIT_KHR`
- *another* `QueueFamily` *might support a-mixture of multiple*
- *talking about this in -> the next part [chapter2.6.]*

```
static inline REY_Array<REY_Array<VkQueueFamilyProperties>>   s_HardwareGPU_QFamProps_List2D;
#define amVK_2D_GPUs_QFAMs                      amVK_Instance::s_HardwareGPU_QFamProps_List2D
    // "REY_LoggerNUtils/REY_Utils.hh" 😁

static inline void GetPhysicalDeviceQueueFamilyProperties(void) {
    amVK_2D_GPUs_QFAMs.reserve(amVK_GPU_List.n);            // malloc using "new" keyword
    for ( uint32_t k = 0;  k < amVK_GPU_List.n; k++ )       // for each GPU
    {
        REY_Array<VkQueueFamilyProperties> *k_QFamProps = &amVK_2D_GPUs_QFAMs.data[k];

        uint32_t queueFamilyCount = 0;
            vkGetPhysicalDeviceQueueFamilyProperties(amVK_GPU_List[k], &queueFamilyCount,
nullptr);

        k_QFamProps->n = queueFamilyCount;
        k_QFamProps->data = new VkQueueFamilyProperties[queueFamilyCount];
            vkGetPhysicalDeviceQueueFamilyProperties(amVK_GPU_List[k], &k_QFamProps->n,
k_QFamProps->data);
    }
}
```

- **Visualization / [See it] / JSON Printing:-** **4.guide.chapter2.5.json.hh**
  - *Check the 3070 JSON by REY*
- **So far, The result:-** **4.guide.chapter2.5.TheEnd.hh**
  - *Compare to ->* *4.guide.chapter2.1.midway.hh*
    - `2DArray_QFAM_Props` *part & below were added only compared to* `Chapter2.1.`

6. `VkQueueFamilyProperties`

- https://vkdoc.net/man/VkQueueFamilyProperties
- **REY_DOCs**
  - `.queueFlags` *-> we are gonna choose a* `QCI.queueFamilyIndex` *based on these flags*
    - *primarily, for the least, we wanna choose a* `QueueFamily` *that supports* `VK_QUEUE_GRAPHICS_BIT`
    - *all kinds of amazing things can be done using*
      - `VK_QUEUE_COMPUTE_BIT`
      - `VK_QUEUE_TRANSFER_BIT`
      - `VK_QUEUE_VIDEO_ENCODE_BIT_KHR`
  - `.queueCount` *-> yes there is a limit to 'how many* `Queues` *we are allowed to work with'* 😂

7. `VkDeviceQCI.queueFamilyIndex`

- `QCI => QueueCreateInfo`
  - *[VkDeviceQueueCreateInfo]*
- **REY_DOCs**
  - `Task:-` *is to choose a* `QueueFamily` *that supports* `VK_QUEUE_GRAPHICS_BIT` 😊
    - *(if you've followed on so far -> this should be easy* 😊*)*
  - *Resolving all of this into* `amVK_Device.hh`

```
void amVK_Device::Select_QFAM_GRAPHICS(void) {
    if (!amVK_Instance::called_GetPhysicalDeviceQueueFamilyProperties) {
        amVK_Instance::EnumeratePhysicalDevices();
    }

    if (!amVK_Instance::called_GetPhysicalDeviceQueueFamilyProperties) {
        amVK_Instance::GetPhysicalDeviceQueueFamilyProperties();
    }

    amVK_Instance::amVK_PhysicalDevice_Index index =
amVK_HEART->GetARandom_PhysicalDevice_amVK_Index();
    this->QCI.Default.queueFamilyIndex =
amVK_Instance::ChooseAQueueFamily(VK_QUEUE_GRAPHICS_BIT, index);
}
```

-- | -- | -- |

---

8. back to `vkCreateDevice()` [finally calling it ☺]

- **REY DOCs**

```
amVK_Device* D = new amVK_Device(amVK_HEART->GetARandom_PhysicalDevice());
    // VkDeviceCreateInfo CI => Class Member
    // VkDeviceQueueCreateInfo QCI => Class Member
D->Select_QFAM_GRAPHICS();
D->CreateDevice();
```

  - *Think of this as a PSeudoCode / or / check out my code if you wanna*
  - `CreateInfo` *=> By default has initial values inside* `amVK_Device`

9. Organizing stuff into classes....

1. `amVK_Props.hh`
   i. `class amVK_Props`
      - `amVK_Instance::GetPhysicalDeviceQueueFamilyProperties()`
      - `amVK_Instance::EnumeratePhysicalDevices()`
      - *& Everything related to those two + The Data + The Properties*

10. `vkGetPhysicalDeviceProperties()`

- https://vkdoc.net/man/vkGetPhysicalDeviceProperties
- `VkPhysicalDeviceProperties` :- https://vkdoc.net/man/VkPhysicalDeviceProperties
  - `.deviceType` :- *https://vkdoc.net/man/VkPhysicalDeviceType*
  - `.limits` :- *save it for later* 😉
  - *you don't need to read the whole documentation of this page*
- for now we won't need, we will need in `ChapterZZZ`