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`

0. `amVK` wrap

```
#include "amVK_Device.hh"

    // TwT
amVK_GlobalProps::EnumeratePhysicalDevices();
amVK_GlobalProps::GetPhysicalDeviceQueueFamilyProperties();

amVK_Device* D = new amVK_Device(amVK_GlobalProps::GetARandom_GPU());
    D->select_QFAM_Graphics();
    D->CI                           // VkDeviceCreateInfo       [public]
    D->QCI.Default                  // VkDeviceQueueCreateInfo  [public]
    D->QCI.Array                    // REY_ArrayDYN<VkQCI>      [public]
        // You can take your own VkDeviceQueueCreateInfo & push_back into this array
    D->CreateDevice();
```

1. `vkEnumeratePhysicalDevices(m_instance, &m_deviceCount, nullptr)`

- https://vkdoc.net/man/vkEnumeratePhysicalDevices
- `</> TheCode`

```
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 physicalDevice` = ⬛ `HardwareGPU_List[0]`
    - *How to 'choose'?* ⬜ *ChapterZZZ*
  - `param pCreateInfo` = ⬛ 🧑‍🎨
  - `param pAllocator` = ⬜ *ChapterZZZ*
  - `param pDevice` = ⬛ `&m_Device`

- *We are not gonna call the* `vkCreateDevice()` *yet....*
  - *But, yes, we've already made the class container around it* 😅
    - *4.guide.chapter2.2.midway.hh*
  - *we'll actually call this functiion in* 🟧 ***Chapter2.8***
  - *Then, Why am I telling you about this now, here?*
    - *because, the idea is, our sole task is to* ***fill it up step by step***
    - *so we did need to know first about* `vkCreateDevice()`
      </br>

- `📽️ So far, The result` :-
  - *4.guide.chapter2.2.midway.hh*

3. `VkDeviceCreateInfo`

- https://vkdoc.net/man/VkDeviceCreateInfo
  - `.sType` = ▨ `VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO`
  - `.pNext` = ▨ `NULL`
    - *lots of interesting Extensions 😉 (will talk about them later)*
    - *Almost any extension that you are gonna need to enable, is probably gonna end up being passed on here too....*
  - `.flags` = ▨ `0`
    - *reserved for future use.*
  - `.pQueueCreateInfos` --> 🔗 *Next SubChapter*
    - *Multiple Queue Create Infos:-* ▨ *Chapter2.8*
  - `.ppEnabledLayerNames` --> *deprecated [by Vulkan]*
  - `.ppEnabledExtensionNames` --> ▨ *Chapter4.2*
  - `.pEnabledFeatures` --> ▨ *ChapterZZZ*
    - *This should be really interesting*
- **📜 REY_DOCs**
  - `.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*
- `📽 So far, The result:-`
  - *4.guide.chapter2.3.midway.hh*

---

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

- https://vkdoc.net/man/VkDeviceQueueCreateInfo
  - `.sType` = ▨ `VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO`
  - `.pNext` = ▨ `NULL`
    - *2 Extensions 😉 (will talk about them later)*
  - `.flags` = ▨ `0`
    - *https://vkdoc.net/man/VkDeviceQueueCreateFlagBits | ivirtex-github*
    - *Only Option:-*
      - ▨ `VK_DEVICE_QUEUE_CREATE_PROTECTED_BIT` *[Protected Queue]*
  - `.queueFamilyIndex` --> 🔗 *Next 3 SubChapters*
  - `.pQueuePriorities` --> *yes, this can be multiple "Priorities"* 😆 *[idk yet why tho]*

- `📽 So far, The result:-`
  - *4.guide.chapter2.4.midway.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 -> 🔗 Next SubChapter*

- `</> TheCode`

```
#define GPUs                                    amVK_GlobalProps::s_HardwareGPU_List
#define amVK_2D_GPUs_QFAMs                         amVK_Instance::s_HardwareGPU_QFamProps_List2D
static inline REY_Array<REY_Array<VkQueueFamilyProperties>>   s_HardwareGPU_QFamProps_List2D;
    // REY_Array --> "REY_LoggerNUtils/REY_Utils.hh" 😄
    // 1 System/PC
        // multiple GPU
            // multiple QFamProps
```

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

        uint32_t QFamCount = 0;
            vkGetPhysicalDeviceQueueFamilyProperties(GPUs[k], &QFamCount, nullptr);

        k_QFamProps->n = QFamCount;
        k_QFamProps->data = new VkQueueFamilyProperties[QFamCount];
            vkGetPhysicalDeviceQueueFamilyProperties(GPUs[k], &k_QFamProps->n, k_QFamProps->data);
    }
    #undef GPUs
}
```

- `👀 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.amVK_Instance.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'* 😆
  - `.timestampValidBits`
  - `.minImageTransferGranularity`

---

7. `VkDeviceQCI.queueFamilyIndex`

- 🎯 `Task`
  - *is to choose a* `QueueFamily` *that supports* `VK_QUEUE_GRAPHICS_BIT` 😉
  - *(if you've followed on so far -> this should be easy* 😉*)*
- `</> 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);
        // If you wanna see the implementation for this function
}
```

  - 📽️ `So far, The result` :-
    - *4.guide.chapter2.9.Props.hh*
    - *4.guide.chapter2.9.amVK.cpp*

---

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

- `</> main.cpp`

```
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. `</> [multiple] VkDeviceCreateInfo.pQueueCreateInfos`

```
/* ============ 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.chapter2.7.TheEnd.hh

---

10. Organizing stuff into classes....

- `amVK_GlobalProps.hh`
  i. `class amVK_GlobalProps`
    - `amVK_Instance::GetPhysicalDeviceQueueFamilyProperties()`
    - `amVK_Instance::EnumeratePhysicalDevices()`
    - *& Everything related to those two + The Data + The Properties*
  - *https://github.com/REYNEP/amGHOST/tree/3e44b982902a3f3fa4ac584aefb19da3d4cdfcc6*
  - `📽 So far, The result` :-
    - *4.guide.chapter2.9.Props.hh*
    - *4.guide.chapter2.9.amVK.cpp*

---

11. `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