REYNEP's Vulkan "Adventure Guide"

Where, you adventure on your own ☺, I only 'guide', showing you the roadmap

Chapter 0: Prerequisites

1. What is `Vulkan`? .... Why `Vulkan`?

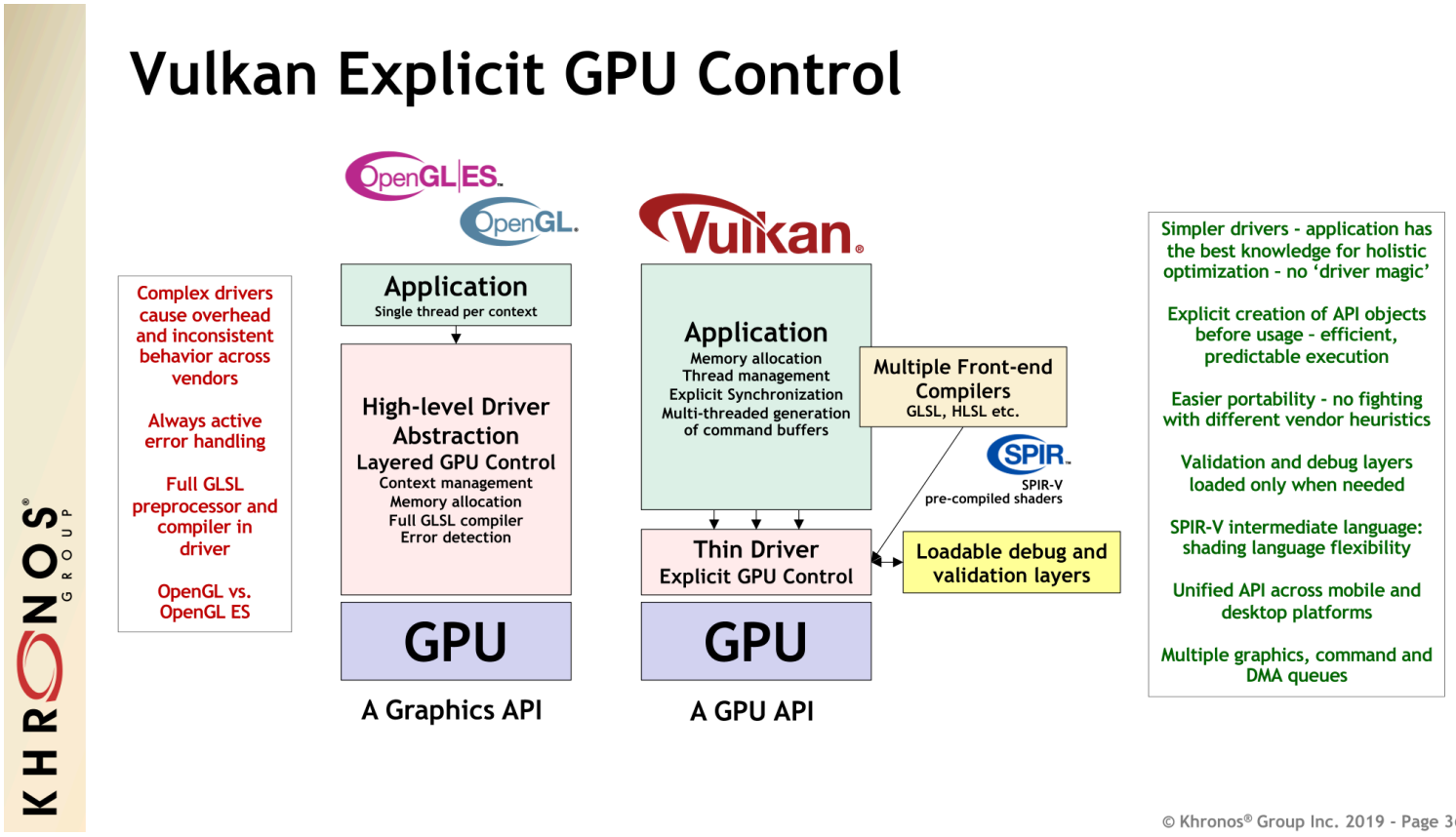
1. Read the `1 - Introduction` part from here only ☺
   i. https://paminerva.github.io/docs/LearnVulkan/01.A-Hello-Window
   ii. *[TODO:-] Convert (above page) to PDF and add a link to that*
2. Alternatively:- you can give this page a try too:- https://vkdoc.net/chapters/fundamentals
3. Why should *'you'* learn/use `Vulkan`?
   i. *Faster*
   ii. *More Control*
   iii. *Lower Level API*
4. Why is this Important?
   i. *Well if you are planning on becoming a game dev, then yeah. Otherwise OpenGL is kinda enough.*
5. When will I need `vulkan`?
   i. *kind of never, unless you've grown tired of OpenGL*
6. How does `vulkan` work?
   - *Rest of the document is dedicated to answer this question* 😊

2. grab `vulkan-sdk`, `cmake`, `amGHOST`

1. https://vulkan.lunarg.com/sdk/home
   - *make sure* `VULKAN_SDK` & `VK_SDK_PATH` *environment variables are set*
   - *restart vscode after installing*
2. https://cmake.org/download/
   - *[optional]* https://enccs.github.io/intro-cmake/hello-cmake/
   - *[optional] OR: Watch 6/7 videos from this playlist:-* https://www.youtube.com/playlist?list=PLK6MXr8gasrGmIiSuVQXpfFuE1uPT615s
   - *restart vscode after installing*
3. if you don't have `vscode` & `C++ Compiler` --> see 4.guide.CH0.vscode.md
4. `git clone -b win32-intro https://github.com/REYNEP/amGHOST`
   - *Open it with VSCode*
   - `F1` --> `CMake: Configure`
   - `F1` --> `CMake: Build`
   - `F1` --> `CMake: Install` --> `.insall` *dir*
   - *check's* ***amGHOST's Usage Example*** *inside* `amGHOST/README.md`
   - `Option 1` *:- use* `cmake` *for your project too.... using* `add_subdirectory(amGHOST)`
   - `Option 2` *:- use* `libamGHOST.lib` *after installing &* `#include amGHOST/<header>`
   - *just copy paste* ***amGHOST's Usage Example*** *into a* `main.cpp` *for your program*
     - *now you shall have a OS-Window* 😊

The Real "Adventure" begins here!

[well, not really. I believe the real adventure is it SHADERs and Algorithms!]



Chapter 1: `VkInstance`

1. `VkApplicationInfo`

- https://vkdoc.net/man/VkApplicationInfo
  - *do remember to check the* `Valid Usage` *section* 😉
- yes, what are you waiting for, go go, shooo....
  i. `#include <vulkan/vulkan.h>`
  ii. *take an instance of that* `Struct` *-> Fill it up [*😉*][have the vkdoc.net as assist]*
- **REY_DOCs**
  - `VkApplicationInfo` *-> holds* `name` *and* `version`*, also the* `lowest Vulkan API version` *Your APP "can run" on. [*clarification needed:- lowest or highest*]*
  - *Also, we can set the* `name` *and* `version` *of the* `engine` *(if any) used to create Your APP. This can help* `vulkan driver implementations` *to perform ad-hoc optimizations.*
    - *e.g. like if a Triple-A [AAA] game used, for say,* `Unreal Engine Version 4.1.smth` *idk* 🤷

- *REFs:- 1. minerva*

2. `VkInstanceCreateInfo`

- https://vkdoc.net/man/VkInstanceCreateInfo
  - *yeah, do remember to check the* `Valid Usage` *section* 😉
  - `.ppEnabledLayerNames` -> *"ChapterZZZ"*
  - `.ppEnabledExtensionNames` -> *Chapter4.2*
  - *Don't hesitate about* `EnabledLayer` & `EnabledExtensions` *right now*
    - *come back and add them when you need to* 😉
- **REY_DOCs**
  - *Nothing that I need to add*
  - *Tho if this section gets big, I will create a separate* `.md` *file for that thingy*

3. `VkInstance m_instance = nullptr;`

- https://vkdoc.net/man/VkInstance
  - *again.... yeah, do remember to check the* `Valid Usage` *section* 😉

4. `vkCreateInstance(CI, &m_instance)`

- https://vkdoc.net/man/vkCreateInstance
  - `Valid Usage` *section.... (yeah, everytime)*

5. Error Handling / Checking / Logging

- check out my `amVK_log.hh`
  - *uses REY_LoggerNUtils inside amGHOST*
  - *has a simple* `stackTracer()` *that i basically stripped from blender3D codebase* 😆

6. The Result

- Check out:- 4.guide.chapter1.hh

7. The Unused ones

1. `vkEnumerateInstanceExtensionProperties()` -> *Chapter4.2*
   - *https://vkdoc.net/man/vkEnumerateInstanceExtensionProperties*
2. `Add_InstanceEXT_ToEnable(const char* extName)` -> *Chapter4.2*
   - *this is a amVK/REY Custom Function*

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`

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

Two Questions I keep on pondering 😗

```
- a) Would this make sense to someone else?
- b) Would this make sense to a 5 year old?
```

Keywords in this file

```
ChapterZZZ => **"ChapterZZZ"** Unknown WIP/TBD Chapter
  KEY_DOCs =>
ChapterZ.Z => **_Chapter1.2_**
```

Chapter 4: `VkSwapchainKHR` 🔰

1. `VkSwapchainCreateInfoKHR` ℹ️

- https://vkdoc.net/man/VkSwapchainCreateInfoKHR
  - `.flags` -> *"ChapterZZZ"*
  - `.surface` -> *next part [Chapter4.2]*
  - *image options -> next part [Chapter4.4]*
    - `.minImageCount` -> 😉
    - `.imageFormat` -> 😉
    - `.imageColorSpace` -> 😉
    - `.imageExtent` -> 😉
    - `.imageArrayLayers`
    - `.imageUsage`
    - `.imageSharingMode` -> `EXCLUSIVE/CONCURRENT` *[Toggle]*
  - `VK_SHARING_MODE_CONCURRENT` -> *"ChapterZZZ"*
    - `.queueFamilyIndexCount` -> *if using, must be* `greated than 1`
    - `.pQueueFamilyIndices`
  - *more image options -> next part*
    - `.preTransform` :- `VkSurfaceTransformFlagBitsKHR`
    - `.compositeAlpha` :- `VkCompositeAlphaFlagBitsKHR`
    - `.presentMode` :- `VkPresentModeKHR`
    - `.clipped` :- `VkBool32`
  - `.oldSwapchain` -> *"ChapterZZZ"*

2. `VkSurfaceKHR` 🏄‍♀️

- https://vkdoc.net/man/VkSurfaceKHR
- https://vkdoc.net/extensions/VK_KHR_surface
  - *Yaaaay, we have reached our first extension to enable*
    - *we need to enable it back in* `vkCreateInstance()` *from Chapter1.2*

1. `vkEnumerateInstanceExtensionProperties()`
   - *https://vkdoc.net/man/vkEnumerateInstanceExtensionProperties*
   - *Implement Exactly like Chapter2.1* 😉
     - `vkEnumeratePhysicalDevices()`
2. `IS_InstanceEXT_Available(const char* extName)`

```
bool amVK_Props::IS_InstanceEXT_Available(const char *extName) {
    for (uint32_t k = 0, lim = amVK_EXT_PROPs.n; k < lim; k++) {
        if (strcmp(amVK_EXT_PROPs[k].extensionName, extName) == 0) {    // <cstring>
            return true;
        }
    }
    return false;
}
```

3. `Add_InstanceEXT_ToEnable(const char* extName)`

```
static inline REY_ArrayDYN<char*> s_Enabled_EXTs = REY_ArrayDYN<char*>(nullptr, 0, 0);
    // It will be automatically allocated, resize, as we keep adding 😉
#include <string.h>
```

```
void amVK_Instance::Add_InstanceEXT_ToEnable(const char* extName)
{
    if (!amVK_Props::called_EnumerateInstanceExtensions) {
         amVK_Props::EnumerateInstanceExtensions();
    }

    if (amVK_Props::IS_InstanceEXT_Available(extName)) {
        char  *dont_lose = new char[strlen(extName)];
        strcpy(dont_lose, extName);

        s_Enabled_EXTs.push_back(dont_lose);

        amVK_Instance::CI.enabledExtensionCount = s_Enabled_EXTs.neXt;
        amVK_Instance::CI.ppEnabledExtensionNames = s_Enabled_EXTs.data;
    }
    else {
        REY_LOG_notfound("Vulkan Extension:- " << extName);
    }
}
```

4. OS Specfic SurfaceEXT & Creating it

```
amVK_Instance::Add_InstanceEXT_ToEnable(amGHOST_System::get_vulkan_os_surface_ext_name());
    // or
amVK_Instance::Add_InstanceEXT_ToEnable("VK_KHR_win32_surface");
    // or some other surface name
```

   i. `VkWin32SurfaceCreateInfoKHR` & `vkCreateWin32SurfaceKHR()`
      - *https://vkdoc.net/man/VkWin32SurfaceCreateInfoKHR*

```
pure-virtual VkSurfaceKHR amGHOST_VkSurfaceKHR_WIN32::create(VkInstance I)
{
    amGHOST_SystemWIN32 *heart_win32 = (amGHOST_SystemWIN32 *) amGHOST_System::heart;
    VkWin32SurfaceCreateInfoKHR CI = {
        .sType = VK_STRUCTURE_TYPE_WIN32_SURFACE_CREATE_INFO_KHR,
        .pNext = NULL,
        .flags = 0,
        .hinstance = heart_win32->_hInstance,
        .hwnd = this->W->m_hwnd
            // W = amGHOST_WindowWIN32
    };

    VkSurfaceKHR S = nullptr;
    VkResult return_code = vkCreateWin32SurfaceKHR(I, &CI, nullptr, &S);
    amVK_return_code_log( "vkCreateWin32SurfaceKHR()" );
    return S;
}
```

   ii. `VkXlibSurfaceCreateInfoKHR` & `vkCreateXlibSurfaceKHR()` 🛠 *[wip]*

   iii. **REY_DOCs**
      - *you can also check* `amGHOST_VkSurfaceKHR::create_surface()` 😉

   iv. **So far, The result:- 4.guide.chapter4.2.TheEnd.hh**
      - *in the end people will just use 1 line*

```
VkSurfaceKHR VK_S = amGHOST_VkSurfaceKHR::create_surface(amG_WindowOBJ,
amVK_Instance::s_vk);
```

3. `Naming Patterns`

- example naming patterns for storing all these data.... cz *it's gonna get overwhelming pretty soon, pretty fast*

1. `Arrays`

```
class amVK_Props {
    public:
        // Array of `HardWare amVK_1D_GPUs` connected to motherboard
    static inline REY_Array<VkPhysicalDevice>                              amVK_1D_GPUs;
    static inline REY_Array<REY_Array<VkQueueFamilyProperties>>            amVK_2D_GPUs_QFAMs;
    static inline REY_Array<VkExtensionProperties>                         amVK_1D_InstanceEXTs;
    static inline REY_ArrayDYN<char*>
amVK_1D_InstanceEXTs_Enabled;
    static inline REY_ArrayDYN<SurfaceInfo>                                amVK_1D_SurfaceInfos;
    static inline REY_Array<REY_Array<VkExtensionProperties>>              amVK_2D_GPUs_EXTs;
        // REY_Array  doesn't allocate any memory by default

    #define amVK_LOOP_GPUs(_var_)           \
        for (uint32_t _var_ = 0, lim = amVK_1D_GPUs.n;             _var_ < lim;  _var_++)
    #define amVK_LOOP_QFAMs(_k_, _var_)     \
        for (uint32_t _var_ = 0, lim = amVK_2D_GPUs_QFAMs[_k_].n;  _var_ < lim;  _var_++)
};
```

2. `ChildrenStructs`

```
class amVK_Props {
    public:
    /**
     * VULKAN-EXT:- `VK_KHR_surface`
        *       IMPL:- `amVK_1D_SurfaceInfos`
        */
    class SurfaceInfo {
        public:
        VkSurfaceKHR S = nullptr;
        SurfaceInfo(void) {}
        SurfaceInfo(VkSurfaceKHR pS) {this-> S = pS;}

                REY_Array<REY_Array<VkSurfaceFormatKHR>>          amVK_2D_GPUs_ImageFMTs;

        bool called_GetPhysicalDeviceSurfaceFormatsKHR = false;
        void        GetPhysicalDeviceSurfaceFormatsKHR(void);  // amVK_2D_GPUs_ImageFMTs
    };
};
```

3. `VkFuncCalls`

```
class amVK_Props {
    public:
    static inline bool called_EnumeratePhysicalDevices = false;
    static inline bool called_GetPhysicalDeviceQueueFamilyProperties = false;
    static inline bool called_EnumerateInstanceExtensions = false;

    public:
```

```
    static void EnumeratePhysicalDevices(void);                         // amVK_1D_GPUs
    static       void GetPhysicalDeviceQueueFamilyProperties(void);     // amVK_2D_GPUs_QFAMs
    static void EnumerateInstanceExtensions(void);                      // amVK_1D_InstanceEXTs
};
```

- **REY_DOCs**
  - *Lots of other nice stuffs are happening inside* `amVK_Props.hh`
- **So far, The result:-**
  - *4.guide.chapter4.3.Props.hh*
  - *4.guide.chapter4.3.Props.cpp*
  - *4.guide.chapter4.3.PropsOLD.hh*

---

4. `SwapChain Image Options` 🖼

1. `vkGetPhysicalDeviceSurfaceFormatsKHR()`
   - *https://vkdoc.net/man/vkGetPhysicalDeviceSurfaceFormatsKHR*
     - `param surface`
   - **REY_DOCs**
     - *Implement Exactly like Chapter2.5* 😉
       - `vkGetPhysicalDeviceQueueFamilyProperties()`
       - *Only difference is,* `Formats` *might be a bit different as per* `VkSurfaceKHR`
2. `VkSurfaceFormatKHR`
   - *https://vkdoc.net/man/VkSurfaceFormatKHR*
   - **REY_DOCs**
     - *Combo of* `ImageFormat` & `ColorSpace`
       - *so, the gpu kinda expects you to respect these combos, instead of mumbo-jumbo-ing & mixing random stufs alltogether....*
       - *altho, even if you do so, gpu is probably gonna show you the result of WRONG COLORSPACE/IMAGEFORMATs on the screen*
3. `Life is Hard without Images/Visualization`
   - *So we are gonna Export to JSON/YAML*
   - *4.guide.chapter4.4.3.Enum2String.hh*
   - *4.guide.chapter4.4.3.data.jsonc*
   - *4.guide.chapter4.4.3.Export.cpp*
     - *dw, don't use this code, it will be refactored & organized in Chapter4.4.6*
4. `VkSurfaceCapabilitiesKHR`
   - *https://vkdoc.net/man/VkSurfaceCapabilitiesKHR*
   - **REY_DOCs**
     - `.minImageCount`
       - *2DriverIMPL:- **must** be at least 1*
     - `.currentExtent`
       - *as the OS Window size changes,* `SurfCaps` *also change*
       - *call* `vkGetPhysicalDeviceSurfaceCapabilitiesKHR()` *to get updated* `WindowSize` / `SurfCaps`
     - `.maxImageArrayLayers`
       - *2DriverIMPL:- **must** be at least 1*
     - `.supportedTransforms`
       - *2DriverIMPL:- at least 1 bit **must** be set.*
     - `.supportedUsageFlags`
       - *2DriverIMPL:-* `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` ***must** be included in the set. Implementations **may** support additional usages.*
     - `.supportedCompositeAlpha`
       - `ALPHA-Blending/Transparency/GlassEffect` *:- you'd have to enable blending/transparency @ OS-Level first, iguess* 🤔
       - *Transparency -> "ChapterZZZ"*
5. `vkGetPhysicalDeviceSurfaceCapabilitiesKHR()`
   - *https://vkdoc.net/man/vkGetPhysicalDeviceSurfaceCapabilitiesKHR*

- **REY_DOCs**
  - *we add on top of Chapter4.4.1* 😉
    - `vkGetPhysicalDeviceSurfaceFormatsKHR()`
  - *4.guide.chapter4.4.5.midway.cpp*

6. `Life is Hard without Images/Visualization 2`
   - *Soooooo many things to keep track of, So here we go again*
   - *4.guide.chapter4.4.6.Export.cpp*
   - *4.guide.chapter4.4.6.data.jsonc*
7. `VkSharingMode`
   - *https://vkdoc.net/man/VkSharingMode*
   - *it's like a Toggle/Button ->* `EXCLUSIVE/CONCURRENT`
8. **So far, The result:-**

```
amVK_SwapChain *SC = new amVK_SwapChain(VK_Surface);
    SC->CI.imageFormat = VK_FORMAT_B8G8R8A8_UNORM;
    SC->CI.imageColorSpace = VK_COLOR_SPACE_SRGB_NONLINEAR_KHR;
    SC->CI.minImageCount    =
amVK_Props::amVK_1D_SurfaceInfos[0].amVK_1D_GPUs_SurfCAP[0].minImageCount;
    SC->CI.imageExtent      =
amVK_Props::amVK_1D_SurfaceInfos[0].amVK_1D_GPUs_SurfCAP[0].currentExtent;
    SC->CI.imageArrayLayers =
amVK_Props::amVK_1D_SurfaceInfos[0].amVK_1D_GPUs_SurfCAP[0].maxImageArrayLayers;
        // You can just use "1" too, which is guranteed by DRIVER_IMPLEMENTATION [2DriverIMPL]
    SC->CI.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
        // `EXCLUSIVE/CONCURRENT` [Toggle]
    SC->CI.imageUsage       = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
        // 2DriverIMPL:- VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT is guranteed to be supported by
SurfCAP
```

9. `Abbreviations`

- `SurfCAP` -> https://vkdoc.net/man/VkSurfaceCapabilitiesKHR
- `SurfFMT` -> https://vkdoc.net/man/VkSurfaceFormatKHR
- `SC` -> SwapChain

10. `VkSwapchainCreateInfoKHR` ℹ [So Far]

- https://vkdoc.net/man/VkSwapchainCreateInfoKHR
  - `.flags` -> *"ChapterZZZ"*
  - `.surface` -> *Chapter4.2* `VkSurfaceKHR` 🐍
  - *image options ->* *Chapter4.4*
    - `.minImageCount` -> 😉 `SurfCAP.minImageCount`
    - `.imageFormat` -> 😉 `SurfFMT[x].format`
    - `.imageColorSpace` -> 😘 `SurfFMT[x].colorSpace`
      - *Choosing a Combo ->* *"ChapterZZZ"*
      - *Compositing & ColorSpaces ->* *"ChapterZZZ"*
    - `.imageExtent` -> 😉 `SurfCAP.minImageCount`
    - `.imageArrayLayers` -> `1`
    - `.imageUsage` -> `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
    - `.imageSharingMode` -> `EXCLUSIVE/CONCURRENT` *[Toggle]*
      - `VK_SHARING_MODE_CONCURRENT` -> *"ChapterZZZ"*
        - *we aren't gonna use concurrent for now*
        - `.queueFamilyIndexCount` -> `0`
        - `.pQueueFamilyIndices` -> `nullptr`

5. `SwapChain Compositing Options` 🤔♂️

1. `.compositeAlpha`
   - *https://vkdoc.net/man/VkCompositeAlphaFlagBitsKHR*
   - **REY_DOCs**
     - `Options` :- *Don't use / Pre-multiplied / Post-multiplied / inherit from OS-native window system*
     - `Requirement` :-
       - *You would have to enable @ OS level first, to enable ALPHA/Transparency/GlassEffect for window-s/surfaces*
       - *then after that, if you query for* `vkGetPhysicalDeviceSurfaceCapabilitiesKHR()`
         - `SurfCAP.supportedCompositeAlpha` *will change*
       - *by default, it's prolly always gonna support*
         - `VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR`
         - *i.e. if you haven't done any mastery wizardry yet, to enable ALPHA/Transparency/GlassEffect*
2. `.preTransform`
   - *https://vkdoc.net/man/VkSurfaceTransformFlagBitsKHR*
   - 🔗 `SurfCAP.currentTransform`
   - *you should probably log it if* `currentTransform` *isn't*
     - `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`
3. `.clipped`
   - *Setting clipped to* `VK_TRUE` *allows the implementation to discard rendering outside of the surface area*
4. `.presentMode` 🏷 `VkPresentModeKHR`
   - *https://vkdoc.net/man/VkPresentModeKHR*
   - **REY_DOCs**
     - `Options` :- *IMMEDIATE / MAILBOX / FirstInFirstOut / FIFO_Relaxed*
5. `.oldSwapChain`
   - *if you are "re-creating" swapchain & you had an oldSwapchain*
   - **REY_DOCs**
     - *We do this when*
       a. *Window Size / WindowExtent / Surface was Changed*

6. `SwapChain Extension Enabling` 🤔