Chapter 6

`amVK_ColorSpace.hh` , `amVK_Surface` , `amVK_SurfacePresenter` , *Renaming Things in amVK*

1. `amVK_ColorSpace.hh`

```
/**
 * ex. 1   amVK_IF::RGBA_8bpc_UNORM
 */
namespace amVK_ImageFormat {
    // 8bpc = 8-bits per channel
    inline constexpr VkFormat RGBA_8bpc_UNORM    = VK_FORMAT_R8G8B8A8_UNORM;    // 37
    inline constexpr VkFormat RGBA_8bpc_SNORM    = VK_FORMAT_R8G8B8A8_SNORM;    // 38
    inline constexpr VkFormat RGBA_8bpc_USCALED  = VK_FORMAT_R8G8B8A8_USCALED;  // 39
    inline constexpr VkFormat RGBA_8bpc_SSCALED  = VK_FORMAT_R8G8B8A8_SSCALED;  // 40
    inline constexpr VkFormat RGBA_8bpc_UINT     = VK_FORMAT_R8G8B8A8_UINT;     // 41
    inline constexpr VkFormat RGBA_8bpc_SINT     = VK_FORMAT_R8G8B8A8_SINT;     // 42
    inline constexpr VkFormat RGBA_8bpc_SRGB     = VK_FORMAT_R8G8B8A8_SRGB;     // 43

    // Common Depth/Stencil Formats
    inline constexpr VkFormat D32_SFLOAT         = VK_FORMAT_D32_SFLOAT;
    inline constexpr VkFormat D24_UNORM_S8_UINT  = VK_FORMAT_D24_UNORM_S8_UINT;
}
#define amVK_IF amVK_ImageFormat
#define amVK_PF amVK_ImageFormat
#define amVK_PixelFormat amVK_ImageFormat
```

- ***Entire Code:-*** *amVK_ ColorSpace.hh*

2. `amVK_Surface`

```
/**
 * VULKAN-EXT:- `VK_KHR_surface`
 *       IMPL:- `amVK_1D_SurfaceInfos`
 */
class amVK_Surface {
  public:
    VkSurfaceKHR S = nullptr;        // Set in CONSTRUCTOR
    amVK_SurfacePresenter *PR = nullptr;    // Set in CONSTRUCTOR

    amVK_Surface(void) {}
    amVK_Surface(VkSurfaceKHR pS);

                REY_Array<REY_Array<VkSurfaceFormatKHR>>              amVK_2D_GPUs_ImageFMTs;
                REY_Array<VkSurfaceCapabilitiesKHR>                   amVK_1D_GPUs_SurfCAP;

    bool called_GetPhysicalDeviceSurfaceFormatsKHR = false;
    bool called_GetPhysicalDeviceSurfaceCapabilitiesKHR = false;
    void        GetPhysicalDeviceSurfaceInfo(void);
    void        GetPhysicalDeviceSurfaceCapabilitiesKHR(void);
};
```

- ***Entire Code:-*** *4.guide.chapter6.3.Surface.hh*

3. `amVK_SurfacePresenter`

```
class amVK_SurfacePresenter {
  public:
    amVK_Surface   *S   = nullptr;
    amVK_SwapChain *SC = nullptr;
    amVK_RenderPass *RP = nullptr;
        //   SC.VkDevice = RP.VkDevice
    amVK_Device        *D = nullptr;
    VkPhysicalDevice   GPU = nullptr;
        // amVK_Device.m_PD = this->GPU;
    amVK_GPU_Index GPU_Index = 0;

  public:
    void bind_Device(amVK_Device *D);
    amVK_SurfacePresenter  (amVK_Surface* pS) {this->S = pS;}

  public:
    amVK_SwapChain*  create_SwapChain(void);
    amVK_RenderPass* create_RenderPass(void);
    // Defined currently inside amVK_SwapChain.cpp

    void                      refresh_SurfCaps(void) { this->S->GetPhysicalDeviceSurfaceCapabilitiesKHR(); }
    VkSurfaceCapabilitiesKHR* fetched_SurfCaps(void) {
        return &( this->S->amVK_1D_GPUs_SurfCAP[this->GPU_Index] );
    }
};
```

- ***Entire Code:-*** *4.guide.chapter6.3.Surface.hh*

4. `amVK` Naming Conventions ☺

1. ***Calling Vulkan Library Functions:-***

```
bool called_GetPhysicalDeviceSurfaceFormatsKHR = false;
bool called_GetPhysicalDeviceSurfaceCapabilitiesKHR = false;
void        GetPhysicalDeviceSurfaceInfo(void);
void        GetPhysicalDeviceSurfaceCapabilitiesKHR(void);
```

2. `vkCreateZZZ()` *wrappers*

```
amVK_SwapChain {
    void CreateSwapChain(void) {
        VkResult return_code = vkCreateSwapchainKHR(this->D->m_device, &CI, nullptr, &this->SC);
        amVK_return_code_log( "vkCreateSwapchainKHR()" );     // above variable "return_code" can nott be
 named smth else
    }
}
```

3. `amVK_Object` */Instance-Creation*

```
amVK_SwapChain* amVK_SurfacePresenter::create_SwapChain(void);
```

4. `amVK_Object::Functions()`

```
amVK_SwapChain*   create_SwapChain(void);           // Creates amVK_Object
amVK_RenderPass*  create_RenderPass(void);          // Creates amVK_Object
void                      refresh_SurfCaps(void);   // SurfCapabilities changes if Window is Resized
VkSurfaceCapabilitiesKHR* fetched_SurfCaps(void);   // Returns the REFRESHED/FETCHED element

void            amVK_SwapChain::sync_SurfCaps(void);/** Refreshes & Syncs `SurfaceCapabilites` */
void            amVK_SwapChain::konf_Images(
    VkFormat IF,
    VkColorSpaceKHR CS,
    VkImageUsageFlagBits IU,
    bool autoFallBack = true
)
void            amVK_SwapChain::konf_Compositing(
    VkPresentModeKHR PM,
    amVK_CompositeClipping CC,
    VkCompositeAlphaFlagBitsKHR CA
);
void            amVK_SwapChain::konf_ImageSharingMode(VkSharingMode ISM);
VkFormat        amVK_SwapChain::active_PixelFormat(void)                    {return CI.imageFormat;}
VkColorSpaceKHR amVK_SwapChain::active_ColorSpace (void)                    {return CI.imageColorSpace;}
```

5. `VkObject` *Variables*

```
class amVK_Image {
  public:
    amVK_Device *D = nullptr;
    VkImage     vk_Image = nullptr;
    VkImageView vk_ImageView = nullptr;
};

class amVK_FrameBuffer {
  public:
    amVK_SurfacePresenter *PR = nullptr;       // Basically, Parent Pointer
    VkFramebuffer vk_FrameBuffer = nullptr;
};

class amVK_RenderPass {
  public:
    amVK_SurfacePresenter *PR = nullptr;       // Basically, Parent Pointer
    VkRenderPass vk_RenderPass = nullptr;
};

class amVK_Surface {
  public:
    amVK_SurfacePresenter *PR = nullptr;   // Created in CONSTRUCTOR
    VkSurfaceKHR vk_SurfaceKHR = nullptr;   //     Set in CONSTRUCTOR
}
```

5. `amVK_RenderPass_Descriptors.hh`

```
namespace amVK_RP_AttachmentDescription
{
        // Change .format before using
    inline VkAttachmentDescription ColorPresentation = {
        .format = VK_FORMAT_UNDEFINED,             // you should use the ImageFormat selected by the swapchain
        .samples = VK_SAMPLE_COUNT_1_BIT,          // We don't use multi sampling in this example
        .loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR,     // Clear this attachment at the start of the render pass
        .storeOp = VK_ATTACHMENT_STORE_OP_STORE,
            // Keep its contents after the render pass is finished (for displaying it)
        .stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE,
            // Similar to loadOp, but for stenciling (we don't use stencil here)
        .stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE,
            // Similar to storeOp, but for stenciling (we don't use stencil here)
        .initialLayout = VK_IMAGE_LAYOUT_UNDEFINED,
            // Layout at render pass start. Initial doesn't matter, so we use undefined
        .finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR,
            // Layout to which the attachment is transitioned when the render pass is finished
            // As we want to present the color attachment, we transition to PRESENT_KHR
    };
};

#define amVK_RPADes amVK_RP_AttachmentDescription
#define amVK_RPARef amVK_RP_AttachmentReference
#define amVK_RPSDes amVK_RP_SubpassDescription
#define amVK_RPSDep amVK_RP_SubpassDependency
```

- *You should kinda check the* `amVK_RenderPass_Descriptors.hh` *file yourself* 😪

```
amVK_RenderPass *RP = PR->create_RenderPass_interface();
    amVK_RPADes::ColorPresentation.format = SC->CI.imageFormat;

    RP->AttachmentInfos .push_back(amVK_RPADes::ColorPresentation);
    RP->SubpassInfos    .push_back(amVK_RPSDes::ColorPresentation);
    RP->Dependencies    .push_back(amVK_RPSDep::ColorPresentation);

    RP->sync_Attachments_Subpasses_Dependencies();
    RP->CreateRenderPass();
```

6. `REY_Utils.hh`

1. `REY_Array`

```
REY_ArrayDYN<VkDeviceQueueCreateInfo> Array = REY_ArrayDYN<VkDeviceQueueCreateInfo>(nullptr, 0, 0);
    // No MemoryAllocation by default 😊
    //      1. REY_ArrayDYN.initialize(10)
    //      2. REY_ARRAY_PUSH_BACK(Array) = your_QueueCI;        [not a function. but rather a preprocessor
macro]
```