GLFW General

GLFWwindow* window;

glfwInit();

glfwWindowHint(GLFW_CLIENT_API, GLFW_NO_API);

window = **glfwCreateWindow**(<WIDTH>, <HEIGHT>, <WINDOW_NAME>, nullptr, nullptr);

glfwSetWindowUserPointer(window, <USER_DATA>);

glfwSetFramebufferSizeCallback(window, <USER_CALLBACK>);

static (void USER_CALLBACK*)(GLFWwindow* window, int width, int height);
// In this callback, just set an atomic flag to recrease the swap chain
// and the other resources.

bool **glfwWindowShouldClose**(window)

glfwGetFramebufferSize(window, &width, &height);

glfwWaitEventsTimeout(<SECONDS>);

glfwDestroyWindow(window);

glfwTerminate();

GLFW Vulkan Specific

int **glfwVulkanSupported**(); // GLFW_TRUE or GLFW_TRUE

int **glfwGetPhysicalDevicePresentationSupport** (instance, VkPhysicalDevice device, uint32_t queuefamily)
// queueFamily : index within the range returned by vkGetPhysicalDeviceQueueFamilyProperties().
// GLFW_TRUE / GLFW_FALSE

const char ** **glfwGetRequiredInstanceExtensions**(uint32_t * count)
// Ex.
// [0] "VK_KHR_surface"
// [1] "VK_KHR_xcb_surface"

VkResult **glfwCreateWindowSurface**(VkInstance instance, window, nullptr, **VkSurfaceKHR * surface**)

GLFWvkproc **glfwGetInstanceProcAddress** (instance, const char * procname)
// typedef void(* GLFWvkproc) (void)
// instance can be nullptr
// procname is something like "vkDestroyImageView"

VkSwapchainCreateInfoKHR createInfo{}
createInfo.surface = surface;
...
vkCreateSwapchainKHR(..., &createInfo, ...);

vkGetPhysicalDeviceSurfaceSupportKHR(..., VkBool32* pSupported)

vkGetPhysicalDeviceSurfaceCapabilitiesKHR( )

```
typedef struct VkSurfaceCapabilitiesKHR {
    uint32_t                         minImageCount; // 2
    uint32_t                         maxImageCount; // 8
    VkExtent2D                       currentExtent;  // (800, 600)
    VkExtent2D                       minImageExtent; // (800, 600)
    VkExtent2D                       maxImageExtent; // (800, 600)
    uint32_t                         maxImageArrayLayers; // 1
    VkSurfaceTransformFlagsKHR       supportedTransforms; // VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR
    VkSurfaceTransformFlagBitsKHR    currentTransform; // VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR
    VkCompositeAlphaFlagsKHR         supportedCompositeAlpha; // VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR
    VkImageUsageFlags                supportedUsageFlags; // VK_IMAGE_USAGE_TRANSFER_SRC_BIT
} VkSurfaceCapabilitiesKHR;                               // VK_IMAGE_USAGE_TRANSFER_DST_BIT
                                                          // VK_IMAGE_USAGE_SAMPLED_BIT
                                                          // VK_IMAGE_USAGE_STORAGE_BIT
                                                          // VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
                                                          // VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
```

vkGetPhysicalDeviceSurfacePresentModesKHR()
// VK_PRESENT_MODE_IMMEDIATE_KHR = 0,
// VK_PRESENT_MODE_FIFO_KHR = 2,
// VK_PRESENT_MODE_FIFO_RELAXED_KHR = 3,

vkGetPhysicalDeviceSurfaceFormatsKHR( )

```
typedef struct VkSurfaceFormatKHR {
    VkFormat           format; // VK_FORMAT_B8G8R8A8_UNORM
                               // VK_FORMAT_B8G8R8A8_SRGB
    VkColorSpaceKHR    colorSpace; // VK_COLOR_SPACE_SRGB_NONLINEAR_KHR
} VkSurfaceFormatKHR;
```