Better Vulkan Application Deployment thank to Vulkan Profiles

A.k.a. “Using the Vulkan Profiles Tools to develop and deploy a Vulkan application”

Christophe Riccio, LunarG

Presentation:
Based on The Vulkan Profiles Tools whitepaper

Agenda

- A Vulkan Profiles introduction
- Creating and Using a Vulkan developer-defined Engine Profile
  - Writing the Engine JSON profile
  - Validating the JSON profile
  - Finding the required Vulkan API version for a profile
  - Generating the Vulkan Profiles API library using the profile
  - Using the Vulkan Profiles API library to check the support of profiles
  - Using the Vulkan Profiles API library to create instances
  - Generating human readable documentation of the profiles
- Creating and Using a Vulkan developer-defined Platform Profile
  - Selecting supported devices
  - Generating the Vulkan platform JSON profile
  - Setup the Profiles layer on the Vulkan developer system
  - Setup the Profiles layer on the C.I. platforms
  - Setup the Profiles layer programmatically
  - Use Vulkaninfo to generate a Device Vulkan profile
A Vulkan Profiles introduction

Why they matter to develop and deploy a Vulkan application?
What are Vulkan Profiles?

- Released with Vulkan 1.3
  - But it’s not really a part of the Vulkan specification, they are essentially developer tools.
- A collection of Vulkan Capabilities
  - Extensions
  - Features
  - Properties
  - Queue properties
  - Formats

- A formalized dialogue method between the applications and the drivers, between components of the Vulkan ecosystem.
Vulkan Profiles use cases:

- **Roadmap profiles**: to express guidance on the future direction of Vulkan devices. Eg: Khronos Roadmap 2024.
- **Platform profiles**: to express the Vulkan support actually available on a platform. Eg: Android Baseline 2021.
- **Device profiles**: to express the Vulkan support of a single Vulkan driver for a Vulkan device. Eg: GPUinfo.org reports
- **Architecture profiles**: to express the Vulkan support of a class of GPUs. Eg: D3D12 Feature Level 12.1
- **Engine profiles**: to express some rendering code paths requirements of an engine. Eg: VP_UE_Vulkan_SM6_RT in Unreal Engine.
- Etc.
Vulkan Profiles by the Vulkan community

- **DXVK**: D3D9 - D3D11 to Vulkan
  - To document the Vulkan driver requirements to run D3D9 - D3D11 applications
  - Eg: VP_DXVK_d3d9_baseline, VP_DXVK_d3d11_level_12_0_optimal

- **vkd3d-proton**: D3D12 to Vulkan
  - To document the Vulkan driver requirements to run D3D12 applications
  - Eg: VP_D3D12_FL_11_0_baseline, VP_D3D12_FL_12_2_optimal, VP_D3D12_maximum_radv

- **Zink**: OpenGL 2.1 - 4.6 to Vulkan
  - To document the Vulkan driver requirements to run OpenGL applications
  - Eg: VP_ZINK_gl21_baseline, VP_ZINK_gl46_optimal

- **Unreal Engine profiles**: To check the Vulkan support of the user system and select the available rendering code paths.
  - Eg: VP_UE_Vulkan_SM5_Android_RT, VP_UE_Vulkan_SM6_RT, VP_UE_Vulkan_SM5
List of the Vulkan Profiles Tools:

- **Vulkan Profiles JSON schema**
  - One JSON schema per Vulkan Header revision to check the correctness of a Vulkan Profiles JSON file

- **Vulkan Profiles file generation**
  - Vulkaninfo and GPUinfo.org export Device Profile JSON files
  - gen_profiles_file.py python script for multiple profiles intersection or union of capabilities
    - VP_LUNARG_desktop_baseline_2022/2023/2024 provided as examples

- **Vulkan Profiles layer**
  - A layer to emulate/clamp profile capabilities on Vulkan developer system

- **Vulkan Profiles API library**
  - Convert JSON files into C++ code
  - A library for Vulkan applications code to check profiles support, to create VkDevice with features enabled
  - A KhronosGroup/Vulkan-Samples sample is available on github for demonstrating the library usage

- **Vulkan Profiles comparison table**
  - Markdown documentation, to easily read, search, compare capabilities across profiles
Vulkan Profiles Tools: How it all comes together?
Creating and Using a Vulkan developer-defined Engine profile
Writing the Engine JSON profile

"$schema": "https://schema.khronos.org/vulkan/profiles-0.8.2-276.json#",
"capabilities": {
    "my_block_name": {
        "extensions":{}, "features":{},
        "properties":{}, "formats":{}
    }
},
"profiles": {
    "VP_LUNARG_example_2024": {
        "version": 1, "api-version": "1.3.204",
        "label": "Vulkan Example 2024 profile",
        "description": "Description of Example 2024 profile",
        "profiles": [ "VP_LUNARG_minimum_requirements_1_3" ],
        "capabilities": [ "my_block_name" ]
    }
}
"capabilities": {
    "my_block_name": {
        "features": {
            "VkPhysicalDeviceFeatures": {
                "multiDrawIndirect": true
            }
        },
        "properties": {
            "VkPhysicalDeviceProperties": {
                "limits": {
                    "maxColorAttachments": 8,
                    "maxBoundDescriptorSets": 7
                }
            }
        }
    }
}
"capabilities": {
    "my_block_name": {
        "formats": {
            "VK_FORMAT_R8G8B8A8_UNORM": {
                "VkFormatProperties": {
                    "linearTilingFeatures": [ "VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT",
                    "VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT", "VK_FORMAT_FEATURE_BLIT_DST_BIT",
                    "VK_FORMAT_FEATURE_TRANSFER_SRC_BIT", "VK_FORMAT_FEATURE_TRANSFER_DST_BIT" ],
                    "optimalTilingFeatures": [ "VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT",
                    "VK_FORMAT_FEATURE_STORAGE_IMAGE_BIT", "VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT",
                    "VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BLEND_BIT", "VK_FORMAT_FEATURE_BLIT_SRC_BIT",
                    "VK_FORMAT_FEATURE_BLIT_DST_BIT", "VK_FORMAT_FEATURE_SAMPLED_IMAGE_FILTER_LINEAR_BIT",
                    "VK_FORMAT_FEATURE_TRANSFER_SRC_BIT", "VK_FORMAT_FEATURE_TRANSFER_DST_BIT" ],
                    "bufferFeatures": []
                }
            }
        }
    }
}
Validating the JSON profile

- To validate Vulkan Profiles file against the schema
  - It can be done online with http://www.jsonschemavalidator.net/
  - It can be done in C++ with libraries such as Valijson
  - It can be done in python with module like jsonschema

- For each Vulkan Header version, we generate a Profiles JSON schema
  - Profiles JSON schemas are available since Vulkan Header 96
    - On Khronos Schema website
    - In Khronos Schema Git repository
Finding the required Vulkan API version for a profile

- Following an example with the Vulkan Roadmap Profiles file
  - Using http://www.jsonschemavalidator.net/
Found 9 error(s)

Message: Property 'VK_KHR_dynamically_rendered_local_read' has not been defined and the schema does not allow additional properties.
Schema path: https://schema.khronos.org/vulkan/profiles-0.8.2-275.json#/properties/capabilities/additionalProperties/properties/extensions/additionalProperties

Message: Property 'VK_KHR_load_store_op_none' has not been defined and the schema does not allow additional properties.
Schema path: https://schema.khronos.org/vulkan/profiles-0.8.2-275.json#/properties/capabilities/additionalProperties/properties/extensions/additionalProperties

…
Schema for Vulkan Header 276


Select schema: Custom

Input JSON:

```json
{
  "$schema": "https://schema.khronos.org/vulkan/profiles-0.8.2-276.json#",
  "capabilities": {
    "vkPhysicalDeviceFeatures": {
      "features": {
        "robustBufferAccess": true
      }
    }
  },
  "vulkan11requirements": {
    "features": {
      "vkPhysicalDeviceVulkan11Features": {
        "multiview": true
      }
    },
    "properties": {
      "vkPhysicalDeviceVulkan11Properties": {
        "maxMultiviewViewCount": 6,
        "maxMultiviewInstanceIndex": 134217727
      }
    }
  }
}
```

 ✓ No errors found. JSON validates against the schema
Generating the Profiles API library using the Engine profile

- The Vulkan SDK ships with the `gen_profiles_solution.py` script
  - To convert Vulkan Profiles from JSON to C++
- This script is used to generate the *Vulkan Profiles API library* with any Profiles needed by the Vulkan application developer
Generating the Profiles API library using the engine profile

python gen_profiles_solution.py
   --registry vk.xml
   --input ./my_profiles/
   --output-library-inc ./my_library/
   --output-library-src ./my_library/
   --debug
Using the Profiles API library to check the support of profiles

VkBool32 supported = VK_FALSE;
VpProfileProperties profile{
    VP_LUNARG_EXAMPLE_2024_NAME, VP_LUNARG_EXAMPLE_2024_SPEC_VERSION};

VkResult result = vpGetInstanceProfileSupport(
    nullptr, &profile, &supported);
if (result != VK_SUCCESS) {
    // something went wrong
    ...
}
else if (supported != VK_TRUE) {
    // profile is not supported at the instance level
    ...
}
Using the Profiles API library to check the support of profiles

```
VkBool32 supported = VK_FALSE;
VpProfileProperties profile{
    VP_LUNARG_EXAMPLE_2024_NAME, VP_LUNARG_EXAMPLE_2024_SPEC_VERSION};

VkResult result = vpGetPhysicalDeviceProfileSupport(
    instance, physicalDevice, &profile, &supported);
if (result != VK_SUCCESS) {
    // something went wrong
    ...
}
else if (supported != VK_TRUE) {
    // profile is not supported at the device level
    ...
}
```
Using the Profiles API library to check the support of profiles

An iterative process to create Engine profiles:

- Hit an assert or validation layer error that check Vulkan requirements in the engine code
- Add these requirements to the Engine profiles file
- Regenerated the library
- The Vulkan application now check correctly the system capabilities on start
Checking Vulkan Profiles variants support

VkResult vpGetInstanceProfileVariantsSupport(
    const char*                     pLayerName,
    const VpProfileProperties*      pProfile,
    VkBool32*                       pSupported,
    uint32_t*                       pPropertyCount,
    VpBlockProperties*              pProperties);

VkResult vpGetPhysicalDeviceProfileVariantsSupport(
    VkInstance                      instance,
    VkPhysicalDevice                physicalDevice,
    const VpProfileProperties*      pProfile,
    VkBool32*                       pSupported,
    uint32_t*                       pPropertyCount,
    VpBlockProperties*              pProperties);
Using the Profiles API library to create instances

VpProfileProperties profile{
   VP_LUNARG_EXAMPLE_2024_NAME, VP_LUNARG_EXAMPLE_2024_SPEC_VERSION};

// Set API version to the minimum API version required by the profile
vkAppInfo.apiVersion = VP_LUNARG_EXAMPLE_2024_MIN_API_VERSION;
VkInstanceCreateInfo vkCreateInfo{ VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO };
vkCreateInfo.pApplicationInfo = &vkAppInfo;
// For additional Vulkan Extensions, add those to vkCreateInfo as usual.
...

VpGetInstanceCreateInfo vpCreateInfo{};
createInfo.pCreateInfo = &vkCreateInfo;
createInfo.enabledFullProfileCount = 1;
createInfo.pEnabledFullProfiles = &profile;

VkInstance instance = VK_NULL_HANDLE;
VkResult result = vpCreateInstance(&vpCreateInfo, nullptr, &instance);
Using the Profiles API library to create instances

VpProfileProperties profile{
    VP_LUNARG_EXAMPLE_2024_NAME, VP_LUNARG_EXAMPLE_2024_SPEC_VERSION};

VkDeviceCreateInfo vkCreateInfo{ VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO };
// For additional Vulkan Extensions and Features, add those to vkCreateInfo
// as usual.
...

VpDeviceCreateInfo vpCreateInfo{};
createInfo.pCreateInfo = &vkCreateInfo;
createInfo.pProfile = &profile;

VkDevice device = VK_NULL_HANDLE;
result = vpCreateDevice(physicalDevice, &vpCreateInfo, nullptr, &device);
## Generating human readable documentation of the profiles

### Vulkan Profiles Definitions

#### Vulkan Profiles List

<table>
<thead>
<tr>
<th>Profiles</th>
<th>VP_KHR_roadmap_2022</th>
<th>VP_ANDROID_baseline_2021</th>
<th>VP_ANDROID_baseline_2022</th>
<th>VP_LUNARG_desktop_baseline_2023</th>
<th>VP_LUNARG_desktop_baseline_2024</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This roadmap profile is intended to be supported by newer devices shipping in 2022 across mainstream smartphone, tablet, laptop, console and desktop devices.</td>
<td>Collection of functionality that is broadly supported on Android</td>
<td>Collection of functionality that is broadly supported on Android</td>
<td>A profile generated by the intersection of a collection of GPUInfo.org device reports to support a large number of actual systems in the Vulkan ecosystem. This profile is meant to be a usage example for Vulkan application developer.</td>
<td>A profile generated by the intersection of a collection of GPUInfo.org device reports to support a large number of actual systems in the Vulkan ecosystem. This profile is meant to be a usage example for Vulkan application developer.</td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Required API version</strong></td>
<td>1.3.204</td>
<td>1.0.68</td>
<td>1.1.106</td>
<td>1.2.148</td>
<td>1.2.197</td>
</tr>
<tr>
<td><strong>Required profiles</strong></td>
<td></td>
<td></td>
<td></td>
<td>VP_LUNARG_minimum_requirements_1_2</td>
<td>VP_LUNARG_minimum_requirements_1_2</td>
</tr>
<tr>
<td><strong>Fallback profiles</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Generating human readable documentation of the profiles

Vulkan Profiles Extensions

- ✓ indicates that the extension is defined in the profile
- "XX Core" indicates that the extension is not defined in the profile but the extension is promoted to the specified core API version that is smaller than or equal to the minimum required API version of the profile
- ✗ indicates that the extension is neither defined in the profile nor is it promoted to a core API version that is smaller than or equal to the minimum required API version of the profile

<table>
<thead>
<tr>
<th>Profiles</th>
<th>VP_KHR_roadmap_2022</th>
<th>VP_ANDROID_baseline_2021</th>
<th>VP_ANDROID_baseline_2022</th>
<th>VP_LUNARG_desktop_baseline_2023</th>
<th>VP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance extensions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_KHR_android_surface</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>VK_KHR_device_group_creation</td>
<td>1.1 Core</td>
<td>✗</td>
<td>✓</td>
<td>1.1 Core</td>
<td>1.1</td>
</tr>
<tr>
<td>VK_KHR_external_fence_capabilities</td>
<td>1.1 Core</td>
<td>✓</td>
<td>✓</td>
<td>1.1 Core</td>
<td>1.1</td>
</tr>
<tr>
<td>VK_KHR_external_memory_capabilities</td>
<td>1.1 Core</td>
<td>✓</td>
<td>✓</td>
<td>1.1 Core</td>
<td>1.1</td>
</tr>
<tr>
<td>VK_KHR_external_semaphore_capabilities</td>
<td>1.1 Core</td>
<td>✓</td>
<td>✓</td>
<td>1.1 Core</td>
<td>1.1</td>
</tr>
<tr>
<td>VK_KHR_get_physical_device_properties2</td>
<td>1.1 Core</td>
<td>✓</td>
<td>✓</td>
<td>1.1 Core</td>
<td>1.1</td>
</tr>
<tr>
<td>VK_KHR_get_surface_capabilities2</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>VK_KHR_surface</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>VK_EXT_swapchain_colorspace</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Device extensions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_KHR_16bit_storage</td>
<td>1.1 Core</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>VK_KHR_8bit_storage</td>
<td>1.2 Core</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>VK_KHR_bind_memory2</td>
<td>1.1 Core</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Generating human readable documentation of the profiles

This table can be generated for any set of profiles using the following command:

```python
python gen_profiles_solution.py
   --registry vk.xml
   --input ./my_engine_profiles/
   --output-doc ./PROFILES.md
```
Creating and Using a Vulkan developer-defined Platform profile
### Selecting supported devices

<table>
<thead>
<tr>
<th>Device</th>
<th>Max. API version</th>
<th>Latest Driver version</th>
<th>Last submission</th>
<th>Count</th>
<th>Compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Corporation Subsystem for Android(TM)</td>
<td>1.3.0</td>
<td>551.23.0.0</td>
<td>2024-01-25 18:46:45</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>NVIDIA GeForce RTX 2080</td>
<td>1.3.271</td>
<td>551.23.0.0</td>
<td>2024-01-25 18:38:03</td>
<td>96</td>
<td>Add</td>
</tr>
<tr>
<td>llvmpipe (LLVM 18.1.0, 256 bits)</td>
<td>1.3.276</td>
<td>0.0.1</td>
<td>2024-01-25 18:35:13</td>
<td>3</td>
<td>Add</td>
</tr>
<tr>
<td>llvmpipe (LLVM 17.0.6, 256 bits)</td>
<td>1.3.276</td>
<td>0.0.1</td>
<td>2024-01-25 18:35:05</td>
<td>20</td>
<td>Add</td>
</tr>
<tr>
<td>llvmpipe (LLVM 16.0.6, 256 bits)</td>
<td>1.3.276</td>
<td>0.0.1</td>
<td>2024-01-25 18:34:58</td>
<td>144</td>
<td>Add</td>
</tr>
<tr>
<td>llvmpipe (LLVM 15.0.7, 256 bits)</td>
<td>1.3.276</td>
<td>0.0.1</td>
<td>2024-01-25 18:34:49</td>
<td>140</td>
<td>Add</td>
</tr>
<tr>
<td>Property</td>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Intel(R) Arc(TM) A750 Graphics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Driver version</td>
<td>101.5186</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>DISCRETE_GPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>API Version</td>
<td>1.3.271</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vendor</td>
<td>INTEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Platform</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Windows</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td>x86_64</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Submitted at</td>
<td>2024-01-25 18:38:14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reportversion</td>
<td>3.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Generating the Vulkan platform JSON profile

```python
python gen_profiles_file.py
   --registry vk.xml
   --input ./VP_LUNARG_desktop_baseline_2024
   --output-path ./VP_LUNARG_desktop_baseline_2024.json
   --output-profile VP_LUNARG_desktop_baseline_2024
   --profile-label "LunarG Desktop Baseline 2024 profile"
   --profile-desc "LunarG Desktop Baseline 2024 description"
   --profile-date 2023-11-01
   --profile-api-version "1.2.197"
   --profile-required-profiles "VP_LUNARG_minimum_requirements_1_2"
   --strip-duplicate-structs
```
Configuring the layers on the developer system

- Based on *Configuring Vulkan Layers whitepaper*
  - Using the GUI application called *Vulkan Configurator*
  - Using environment variables
  - Using the Vulkan API: `vkCreateInstance()` and `VK_EXT_layer_settings`

- The layer settings are documented by each layer:
  - [Profiles layer documentation](#)
  - [Validation layer documentation](#)
Vulkan Configurator 2.5.5-20240115 <ACTIVE>

Vulkan Layers Management
- Layers Fully Controlled by the Vulkan Applications
- Overriding Layers by the Vulkan Configurator
  - Apply only to the Vulkan Applications List
  - Continue Overriding Layers on Exit

Vulkan Layers Configurations
- API dump
- Frame Capture
- Portability
- Synchronization
- Validation

Vulkan Application Launcher
- Application: vkcube
- Clear log at launch
- Vulkan Loader Messages: none

Vulkan Development Status:
- Layers override: "Portability" configuration
- VULKAN_SDK environment variable: E:\VulkanSDK\1.3.275.0-beta
- Vulkan Loader version: 1.3.259
- User-Defined Layers locations:
  - VK_LAYER_PATH variable: None
  - Per-configuration paths: None
  - VK_ADD_LAYER_PATH variable: None
- 'vk_layer_settings.txt' uses the default platform path:
  C:\Users\Piranha\AppData\Local\LunarG\vkconfig\override
- Available Layers:
  - VK_LAYER_NV_optimus
  - VK_LAYER_RENDERDOC_Capture

Portability Settings
- VK_LAYER_KHRONOS_validation
  - Standard Preset
    - Validation Areas
    - Debug Action
      - Log Message
        - Log Filename: stdout
    - Message Severity
      - Info
      - Warning
      - Performance
      - Error
    - Limit Duplicated Messages
      - Max Duplicated Messages: 10
      - Mute Message VUIDs
  - VK_LAYER_KHRONOS_profiles
    - Emulate a Vulkan Portability Profile Preset
      - Force Device (BETA) Off
    - Emulate a Vulkan Profile
      - Profiles Directories
        - P:\LUNARG-desktop-baseline-2023
    - Schema Validation
    - Simulate Profile Capabilities
Configuring the layers for C.I.

Override the layers configuration on the system:

$ vkconfig layers --override configuration-file.json

Stop overriding the layers configuration on the system:

$ vkconfig layers --surrender
Configuring the layers on the C.I. platforms

Enabling and ordering the Vulkan Layers with environment variables:

C:\> set VK_INSTANCE_LAYERS=VK_LAYER_KHRONOS_validation;VK_LAYER_KHRONOS_profiles

Stop overriding the layers configuration on the system:

C:\> set VK_KHRONOS_VALIDATION_VALIDATE_SYNC=true
C:\> set VK_KHRONOS_VALIDATION_DUPLICATE_MESSAGE_LIMIT=3
C:\> set VK_KHRONOS_PROFILES_PROFILE_DIRS=$VULKAN_SDK/Config/VK_LAYER_KHRONOS_profiles
C:\> set VK_KHRONOS_PROFILES_PROFILE_NAME=VP_LUNARG_desktop_baseline_2024
C:\> set VK_KHRONOS_PROFILES_FORCE_DEVICE_UUID=8680A156080000000E00000000000000

- Profiles layer documentation
- Validation layer documentation
Configuring the layers programmatically

- Using `vkCreateInstance` API
- Using the `VK_EXT_layer_settings` extension
const char* val_name = "VK_LAYER_KHRONOS_validation";
const char* pfl_name = "VK_LAYER_KHRONOS_profiles";

const char* setting_profile_dirs[] = {
    "$VULKAN_SDK/Config/VK_LAYER_KHRONOS_profiles"};;
const char* setting_profile_name[] = {
    "VP_LUNARG_desktop_baseline_2024"};;
const VkBool32 setting_thread_safety = VK_TRUE;
const char* setting_debug_action[] = {
    "VK_DBG_LAYER_ACTION_LOG_MSG"};;
const char* setting_report_flags[] = {
    "info", "warn", "perf", "error", "debug"};;

const VkLayerSettingEXT settings[] = {
    {pfl_name, "profile_dirs", VK_LAYER_SETTING_TYPE_STRING_EXT, 1, &setting_profile_dirs},
    {pfl_name, "profile_name", VK_LAYER_SETTING_TYPE_STRING_EXT, 1, &setting_profile_name},
    {val_name, "thread_safety", VK_LAYER_SETTING_TYPE_BOOL32_EXT, 1, &setting_thread_safety},
    {val_name, "debug_action", VK_LAYER_SETTING_TYPE_STRING_EXT, 1, setting_debug_action},
    {val_name,  "report_flags", VK_LAYER_SETTING_TYPE_STRING_EXT, static_cast<uint32_t>(std::size(setting_report_flags)), setting_report_flags}

const VkLayerSettingsCreateInfoEXT layer_settings_create_info = {
    VK_STRUCTURE_TYPE_LAYER_SETTINGS_CREATE_INFO_EXT, nullptr,
    static_cast<uint32_t>(std::size(settings)), settings};
const VkApplicationInfo app_info = initAppInfo();

const char* layers[] = {
  "VK_LAYER_KHRONOS_validation",
  "VK_LAYER_KHRONOS_profiles"
};

const char* extensions[] = {
  "VK_EXT_layer_settings"
};

const VkInstanceCreateInfo inst_create_info = {
  VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO,
  &layer_settings_create_info,
  0,
  &app_info,
  static_cast<uint32_t>(std::size(layers)),
  layers,
  static_cast<uint32_t>(std::size(extensions)),
  extensions
};

VkInstance instance = VK_NULL_HANDLE;
VkResult result = vkCreateInstance(&inst_create_info, nullptr, &instance);
Using Vulkaninfo to generate Device profiles

Useful for the Vulkan application developer to know on what platform the C.I. was running:

```
$ vulkaninfo --json -o ci_instance_with_native_capabilities_profile.json
$ test_runs.sh -o native_capabilities_test_results.txt
$ vkconfig layers --override configuration-file.json
$ vulkaninfo --json -o ci_instance_with_platform_capabilities_profile.json
$ test_runs.sh -o platform_capabilities_test_results.txt
```
Vulkan® Help Us Improve the Vulkan SDK and Ecosystem
Share Your Feedback
Take the LunarG annual developer’s survey
https://www.surveymonkey.com/r/KTBZDCM

- Survey results are tabulated
- Shared with the Vulkan Working Group
- Actions are assigned
- Results are reported

Survey closes February 26, 2024

Thank you!

QUESTIONS?
christophe@lunarg.com

Today’s Presentation:
https://bit.ly/3SkZZ1e

Get A FREE Tumbler at the LunarG Sponsor Table!