Creating Portable Vulkan Applications Using DevSim

Using the Device Simulation Layer to target a hardware ecosystem

Jeremy Kniager, Christophe Riccio, LunarG
September 2021

Introduction

The Vulkan API aims to be portable and lightweight, allowing developers to write applications that can be used on multiple platforms and run efficiently. To implement these two goals, Vulkan provides functions and structs to query data about the underlying platform on which the application is running, and leaves it up to the developer to write a correct application that works within the platform's limitations.

The validation layer assists developers in checking their code for improper use of Vulkan, but these checks take into account only the limits of the test platform. To ensure an application properly handles multiple platforms, these checks must be run on all platforms of interest to the developer. Combinations of GPUs, ICDs, drivers, and operating systems to name a few factors create an exponential number of possible test platforms, which is infeasible for a developer to obtain and maintain.

The Device Simulation Layer (also called Devsim) seeks to mitigate this obstacle by providing a method to simulate arbitrary property, feature, and limit combinations representing different platforms for specific test cases. For example, it can be configured as though the application is running on a device with the minimum properties, features, and limits defined by the Vulkan specification.

Devsim does this by modifying the query data provided by the Vulkan driver to the Vulkan application by the query functions to simulate more restrictive capabilities to ensure the application properly handles reported capabilities.

Some helpful Devsim features that will be discussed throughout this document are:

1. Simulation vs. Emulation
2. Basic Operation of the Devsim Layer
3. Premade configuration files:
    a. Specification minimum: Minimum properties, limits, and features based on the Vulkan 1.0, 1.1, and 1.2 specifications
    b. Portability minimum: Minimum properties, limits, and features for portability subset devices based on the Vulkan 1.0, 1.1, and 1.2 specifications.
    c. Desktop portability: Minimum properties, limits, and features based on what is effectively found in actual hardware for Vulkan 1.0 and 1.2.
4. Improved integration of JSON files from gpuinfo.org
5. Added support for Vulkan 1.1 and 1.2 core capabilities
6. Support for surface query modification:
    a. Surface capabilities
    b. Surface formats
    c. Present modes

7. Emulation for the `VK_KHR_portability_subset` extension
8. Array combination modes for better modification of queries that return arrays

Simulation vs. Emulation

Devsim is shorthand for “Device Simulation Layer.” The primary function of Devsim is simply to simulate the limits, properties, and features of a device, i.e. modifying device responses to query function calls by the application. Of course, the underlying device or driver function are never actually changed, they merely appear to have the limits, properties, and features of a different device or driver.

This is different from emulation that would change the actual behavior of the underlying device or driver to match that of a different device or driver.

In all but one case, Devsim simulates changes and leaves it up to the validation layer to inform the developer about functions that do not adhere to the proper limits.

The one exception is portability subset extension emulation, which causes Devsim to add the `VK_KHR_portability_subset` extension to the device extensions list, and pre-populate the `VkPhysicalDevicePortabilitySubsetPropertiesKHR` and `VkPhysicalDevicePortabilitySubsetFeaturesKHR` structs provided by said extension with default values.

Basic Operation

Before using Devsim, we must have a config JSON file on our computer to represent the Vulkan capabilities we wish to simulate.

The Vulkan SDK contains sample and minimum config files, which are good starting points when testing applications with Devsim.

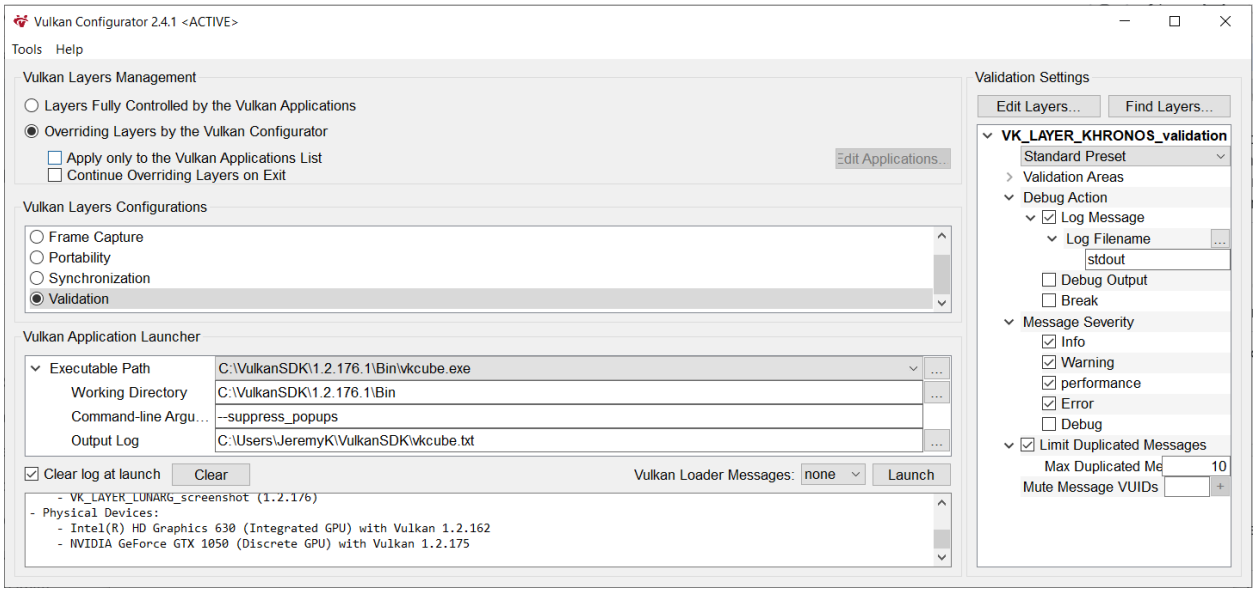
Files representing real world devices can be downloaded from https://vulkan.gpuinfo.org/, which is a database maintained by Sascha Willems.

There are two methods for enabling the layer: 1) use of the Vulkan Configurator and 2) use of the command line.

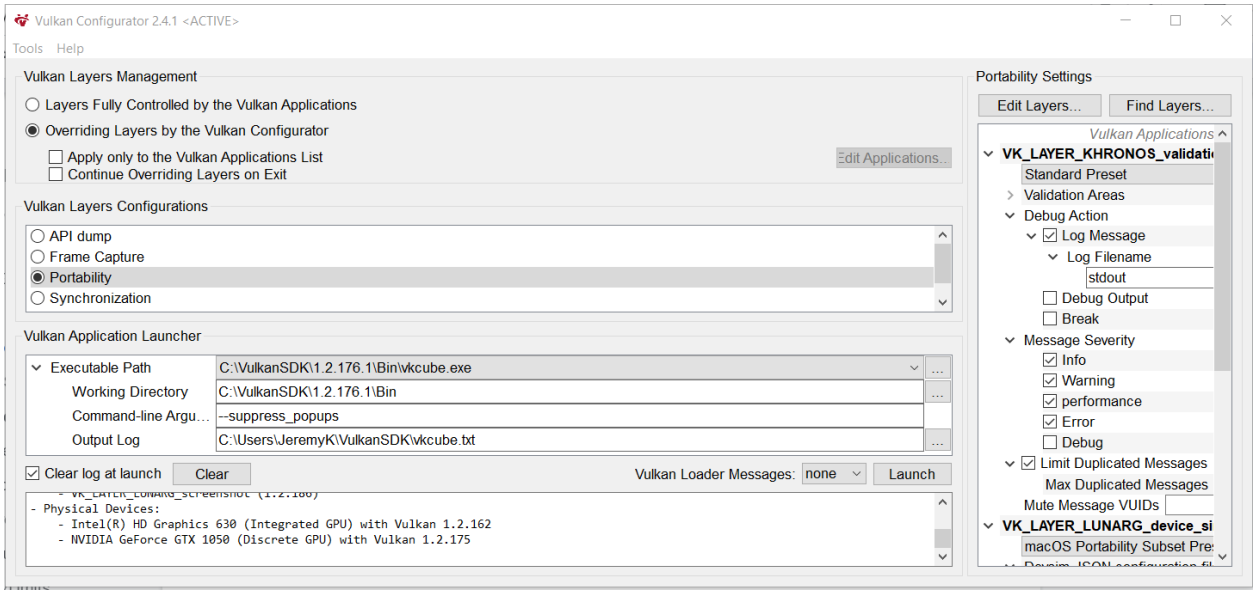
Enabling Devsim using Vulkan Configurator

For Vulkan desktop developers convenience, it is highly recommended to use Vulkan Configurator when setting up layers for your system, especially layers with many different settings like the Devsim layer. VkConfig (Vulkan Configurator) is included in the Vulkan SDK.

When you open the VkConfig application, you will be greeted with the following window.

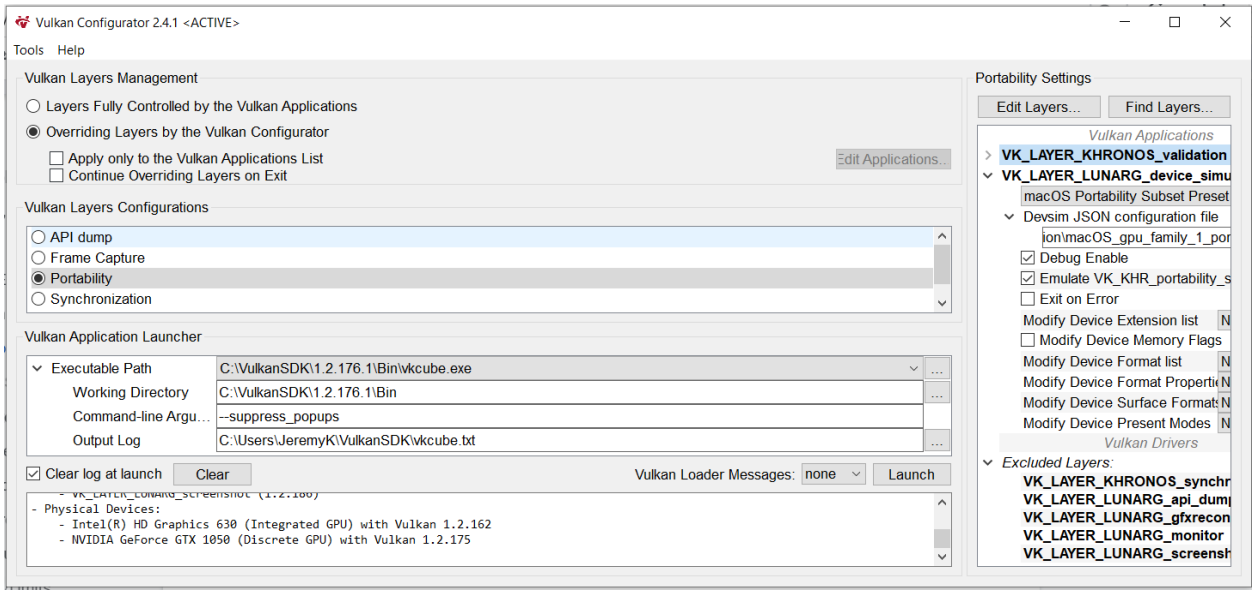


Select the “Portability” built-in configuration from the “Vulkan Layers Configurations” list.

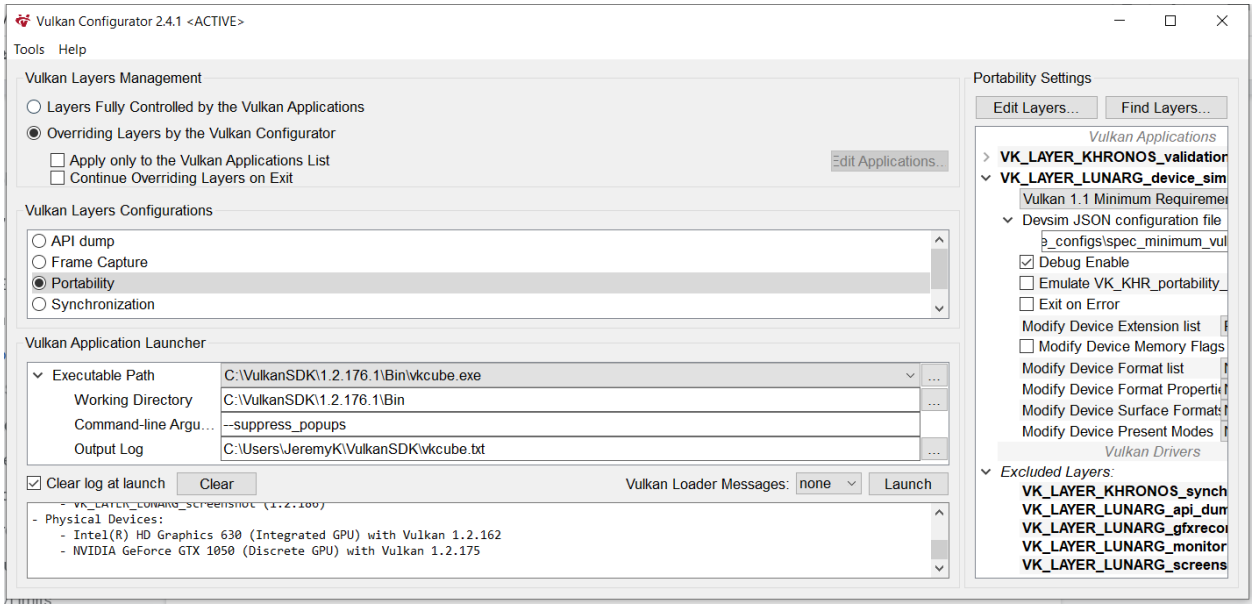


This configuration includes the Khronos Validation Layer and the Device Simulation layer. To the right, we can see the layer settings.

Hide the Khronos Validation Layer setting for now by clicking the carrot next to `VK_LAYER_KHRONOS_validation`.



Now, copy-paste the path of your config file into the “Devsim JSON configuration file” textbox and uncheck the “Emulate VK_KHR_portability_subset” option.



You should now be able to run your Vulkan application with Devsim active.

For more information on VkConfig, [click here to refer to the LunarG documentation](https://github.com/LunarG/VulkanTools/blob/master/vkconfig/README.md) (https://github.com/LunarG/VulkanTools/blob/master/vkconfig/README.md).

Command Line Section

The Vulkan loader has environment variables for enabling layers. These variables are `VK_LAYER_PATH` which is used to manually set the loader's search path for layers, and `VK_INSTANCE_LAYERS` which is used to set active layers and in what order they should be called.

The Devsim layer's name is `VK_LAYER_LUNARG_device_simulation`. When turning it on, make sure it runs closest to the driver, since we want all layers and applications to see the simulated limits instead of the real limits.

Set the `VK_INSTANCE_LAYERS` environment variable to: `VK_INSTANCE_LAYERS=<other layers>:VK_LAYER_LUNARG_device_simulation`.

Next, we need to point Devsim to our desired JSON config file.

You can point to the desired config file with the `VK_LUNARG_DEVICE_SIMULATION_FILENAME` environment variable by setting it to the path of the JSON config file.

From here, we can run our application and get the simulated values.

Config File Anatomy

Basic Config File Design

Devsim config files use the JSON format to store data about a simulated device. In the past, these values were strictly defined by JSON schemas hosted at https://schema.khronos.org/vulkan/. The use of these schemas is rather clunky and awkward, requiring multiple files to describe different data for the same device.

Recently, Devsim was updated to attempt to read all supported structs from any file using a valid schema, allowing all device capabilities to be defined in a single file.

Special cases are also made for the format used by JSON files found at gpuinfo.org.

Config File Heading

The only required property in a Devsim config file is the "$schema" property which tells the layer that the file is meant to be a Devsim config file. Though it can be set to any of the schemas at https://schema.khronos.org/vulkan/ it is recommended that the value be set to

"https://schema.khronos.org/vulkan/devsim_1_0_0.json#" as all other schemas have been deprecated.

Devsim also allows for comments in the file using the "comments" property. This property is defined as a JSON "object" in the schema, and can be filled with whatever properties the user desires.

Example:

```
{
	"$schema": "https://schema.khronos.org/vulkan/devsim_1_0_0.json#"
	"comments": {
		"title": "Example Devsim config file",
		"desc": "This file serves as an example for comments in a Devsim config file."
		"version": 1
	}
}
```

Property, Feature, and Limit Data

The following properties are used to set property, feature, and limit data for the simulated device represented by the config file. Brackets "[ ]" represent an optional portion of the property as Devsim will read in the data with or without that part of the struct name.

Individual Structs

Properties/Limits

- VkPhysicalDeviceProperties
- VkPhysicalDeviceDepthStencilResolveProperties[KHR]
- VkPhysicalDeviceSubgroupProperties
- VkPhysicalDeviceDescriptorIndexingProperties[EXT]
- VkPhysicalDeviceFloatControlsProperties[KHR]
- VkPhysicalDeviceHostQueryResetFeatures[EXT]
- VkPhysicalDeviceMaintenance3Properties[KHR]
- VkPhysicalDeviceMultiviewProperties[KHR]
- VkPhysicalDevicePointClippingProperties[KHR]
- VkPhysicalDevicePortabilitySubsetPropertiesKHR
- VkPhysicalDeviceProtectedMemoryProperties
- VkPhysicalDeviceTimelineSemaphoreProperties[KHR]
- VkPhysicalDeviceMemoryProperties
- VkSurfaceCapabilitiesKHR

Features

- VkPhysicalDeviceFeatures
- VkPhysicalDevice16BitStorageFeatures[KHR]
- VkPhysicalDevice8BitStorageFeatures[KHR]
- VkPhysicalDeviceBufferDeviceAddressFeatures[KHR]
- VkPhysicalDeviceDescriptorIndexingFeatures[EXT]
- VkPhysicalDeviceImagelessFramebufferFeatures[KHR]
- VkPhysicalDeviceMultiviewFeatures[KHR]
- VkPhysicalDevicePortabilitySubsetFeaturesKHR
- VkPhysicalDeviceProtectedMemoryFeatures
- VkPhysicalDeviceSamplerFilterMinmaxProperties[EXT]
- VkPhysicalDeviceSamplerYcbcrConversionFeatures[KHR]
- VkPhysicalDeviceScalarBlockLayoutFeatures[EXT]
- VkPhysicalDeviceSeparateDepthStencilLayoutsFeatures[KHR]
- VkPhysicalDeviceShaderAtomicInt64Features[KHR]
- VkPhysicalDeviceShaderDrawParametersFeatures
- VkPhysicalDeviceShaderFloat16Int8Features[KHR]
- VkPhysicalDeviceShaderSubgroupExtendedTypesFeatures[KHR]
- VkPhysicalDeviceTimelineSemaphoreFeatures[KHR]
- VkPhysicalDeviceUniformBufferStandardLayoutFeatures[KHR]
- VkPhysicalDeviceVariablePointersFeatures[KHR]
- VkPhysicalDeviceVulkanMemoryModelFeatures[KHR]

Arrays of Structs

These properties are arrays that contain structs instead of individual structs.

- ArrayOfVkQueueFamilyProperties
- ArrayOfVkFormatProperties
- ArrayOfVkExtensionProperties
- ArrayOfVkSurfaceFormats
- ArrayOfVkPresentModes

Special Properties

These are structs that are not directly linked to any struct defined in the spec, but contain Vulkan properties, features, and limits that are defined by the spec. Some of these structs are also part of the format at gpuinfo.org.

- Vulkan12Properties
- Vulkan12Features
- core11
    - features: Equivalent to a VkPhysicalDeviceVulkan11Features struct
    - properties: Equivalent to a VkPhysicalDeviceVulkan11Properties struct
- core12

    - features: Equivalent to a VkPhysicalDeviceVulkan12Features struct
    - properties: Equivalent to a VkPhysicalDeviceVulkan12Properties struct
- surfacecapabilites: Please note the missing “i” at the end of “capabilities.” This is how the property is spelled in the GPUinfo format.
    - surfaceformats: Equivalent to ArrayOfVkSurfaceFormats
    - presentmodes: Equivalent to ArrayOfVkPresentModes

For more complete examples of Devsim config files, please refer to the sample files at https://github.com/LunarG/VulkanTools/tree/master/layersvt/device_simulation_examples/sdk_sample_configs.

Devsim Settings

Devsim JSON configuration file

This setting lets the user set the path to one or more config files to provide Devsim with simulation limits. To set multiple config files, provide a string of paths each separated by the system separator. Later files in the list override the values of former files in the list.

By default this setting is an empty string which will cause Devsim to output an error if it is not set to a real file.

Setting Key: `filename`

Using a vk_layer_settings.txt file, the separator is a comma(,):
Example:
```
lunarg_device_simulation.filename = config_1.json,config_2.json,config_3.json
```

Environment Variable: `VK_LUNARG_DEVICE_SIMULATION_FILENAME`

On Unix based systems, such as Linux or macOS, the separator is a colon(:) :
Example:
```
VK_DEVSIM_FILENAME=config_1.json:config_2.json:config_3.json
```

On Windows systems, the separator is a semicolon(;) :
Example:
```
VK_DEVSIM_FILENAME=config_1.json;config_2.json;config_3.json
```

Debug Enable

This boolean setting turns on debug output for Devsim and outputs to stdout.

Off by default.

Setting Key: `debug_enable`

Environment Variable: `VK_LUNARG_DEVICE_SIMULATION_DEBUG_ENABLE`

Exit on Error

This boolean setting will cause Devsim to automatically close the application if it runs into an error. Note that Devsim will still print error messages if this setting is off.

Off by default.

Setting Key: `exit_on_error`

Environment Variable: `VK_LUNARG_DEVICE_SIMULATION_EXIT_ON_ERROR`

Emulate VK_KHR_portability_subset

This boolean setting turns on emulation of the `VK_KHR_portability_subset` extension. This will cause Devsim to add the extension to the device extension list and fill out default values in the extension's structs `VkPhysicalDevicePortabilitySubsetPropertiesKHR` and `VkPhysicalDevicePortabiliySubsetFeaturesKHR`.

On devices and drivers that natively implement the portability subset extension, this setting will do nothing.
Please read the following links for more information about the `VK_KHR_portability_subset` extension and the Vulkan Portability Initiative:

- https://www.khronos.org/assets/uploads/apis/Vulkan-Portability-Update_Jan21.pdf
- https://www.lunarg.com/wp-content/uploads/2021/06/The-State-of-Vulkan-on-Apple-03June-2021.pdf

Off by default.Setting Key: `emulate_portability`

Environment Variable:
`VK_LUNARG_DEVICE_SIMULATION_EMULATE_PORTABILITY`

Modify Device Memory Flags

Devsim will not modify memory flags by default. This boolean value turns on memory flag modification in case a user needs to perform a test that requires it.

Off by default.

Setting Key: `modify_memory_flags`

Environment Variable:
`VK_LUNARG_DEVICE_SIMULATION_MODIFY_MEMORY_FLAGS`

Modify Device Extension list

An Array Combination Mode setting that modifies the array of `VkExtensionProperties` returned by `vkEnumerateDeviceExtensionProperties`.

Setting Key: `modify_extension_list`

Environment Variable:
`VK_LUNARG_DEVICE_SIMULATION_MODIFY_EXTENSION_LIST`

Format Modification

Two settings are used to set how format queries are modified:

Modify Device Format list

An Array Combination Mode setting that modifies which formats will be treated as supported by the `vkGetPhysicalDeviceFormatProperties`.

Setting Key: `modify_format_list`

Environment Variable: `VK_LUNARG_DEVICE_SIMULATION_MODIFY_FORMAT_LIST`

If a format is unsupported, the function will return a VkFormatProperties struct with all variables set to 0. If a format is set to unsupported by the config file and this setting, then devsim will modify the VkFormatProperties struct to the unsupported state.

Modify Device Format Properties

An Array Combination Mode setting that modifies the property flags within the VkFormatProperties struct for each supported format.

Setting Key: modify_format_properties

Environment Variable: VK_LUNARG_DEVICE_SIMULATION_MODIFY_FORMAT_PROPERTIES

Whereas the VK_DEVSIM_MODIFY_FORMAT_LIST setting will 0 out VkFormatProperties structs completely, this setting allows the user finer control over which properties are supported by each format.

Note: If a format is marked unsupported by the VK_DEVSIM_MODIFY_FORMAT_LIST, then it will be set to the unsupported state whether this setting would give it properties or not.

Example: VK_DEVSIM_MODIFY_FORMAT_LIST is set to “blacklist”. VK_FORMAT_R8B8G8A8 is included in the ArrayOfVkFormatProperties with optimalTilingFeatures set to the VK_FORMAT_FEATURE_SAMPLED_IMAGE_BIT and the VK_FORMAT_FEATURE_COLOR_ATTACHMENT_BIT. Even if VK_DEVSIM_MODIFY_FORMAT_PROPERTIES was set to “replace”, calling vkGetPhysicalDeviceFormatProperties on VK_FORMAT_R8G8B8A8 would return a VkFormatProperties struct in the unsupported state.

Modify Device Surface Formats

An Array Combination Mode setting that modifies the array of surface formats returned by vkGetPhysicalDeviceSurfaceFormatsKHR. Surface formats are considered equivalent ONLY if their format properties AND colorSpace properties are equal to each other.

Setting Key: `modify_surface_formats`

Environment Variable:
`VK_LUNARG_DEVICE_SIMULATION_MODIFY_SURFACE_FORMATS`

Modify Device Present Modes

An Array Combination Mode setting that modifies the array of `VkPresentModeKHR` structs that is returned by `vkGetPhysicalDevicePresentModesKHR`.

Setting Key: `modify_present_modes`

Environment Variable:
`VK_LUNARG_DEVICE_SIMULATION_MODIFY_PRESENT_MODES`

Array Combination Modes

Some queries to the driver or device return an array of values. Previously to handle these, Devsim would require the config file to provide a full list of entries to be simulated. This made simulating specific test cases difficult. To improve the configuration of these arrays of values Devsim now includes a special category of settings known as Array Combination Mode Settings.

Array Combination Mode Settings are enum settings with five different valid values:

- “none”: This is the default value for settings in this category. It turns off all modification of the queries and just uses the real values.
- “replace”: This value handles queried arrays in the original way. It turns on full replacement of the queried array.
- “whitelist”: This value replaces the queried array with values from the config file only if those values are included in the original array.
- “blacklist”: This value removes entries from the real array if they are in the config file. This could be used to test if the application properly handles situations where it does not have access to certain extensions or features.
- “intersect”: This value adds the entries from the config file to the real array while avoiding duplicates.

The following settings all take in Array Combination Mode Enums.

All Array Combination Mode settings are “none” by default.

Summary

In its design for cross-platform portability, the Vulkan API includes many checks for platform capabilities that developers must consider and test when developing Vulkan applications. Devsim is a powerful and versatile layer that helps developers test and debug their applications without having to spend valuable resources creating every platform/capability test combination.