Configuring Vulkan Layers

A consistent approach to configure layers

Christophe Riccio, LunarG
April 2024

Configuring Vulkan Layers approaches

Vulkan supports intercepting or hooking API entry points via a layer framework. A layer can intercept all or any subset of Vulkan API entry points. Multiple layers can be chained together to cascade their functionality in the appearance of a single, larger layer.

Vulkan layers allow application developers to add functionality to Vulkan applications without modifying the application itself, e.g., validating API usages, dumping API entry points or generating screenshots of specified frames.

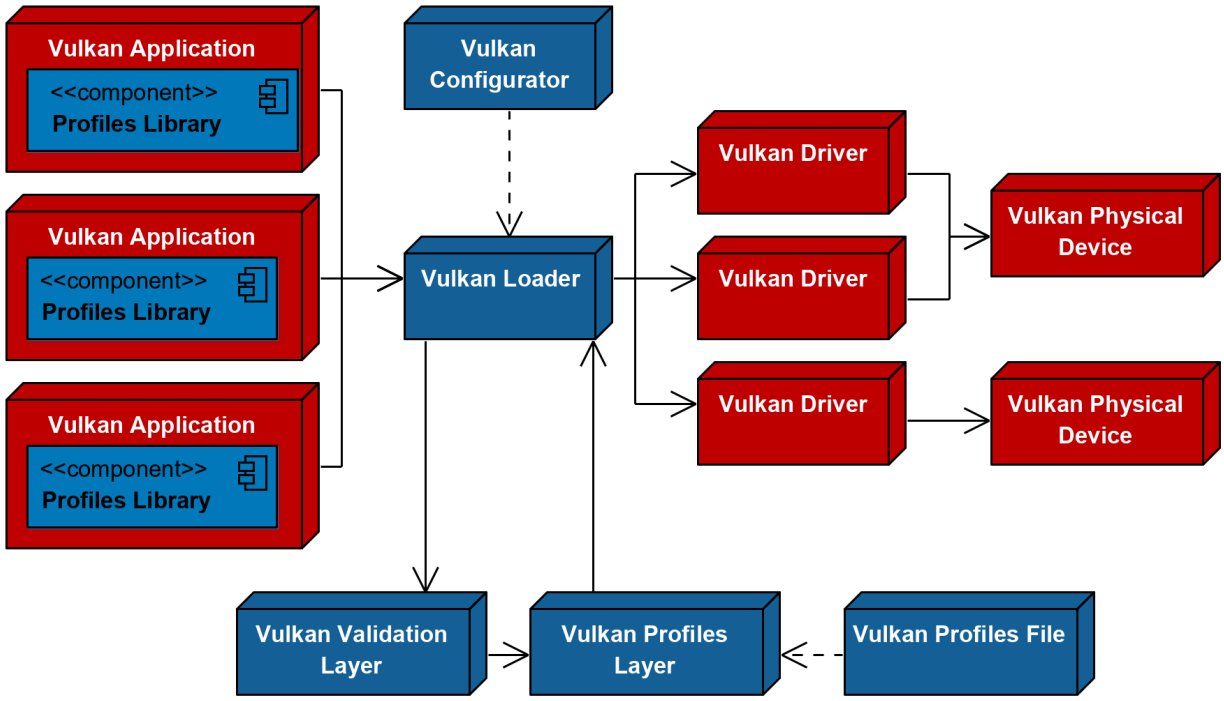
Vulkan layers can be configured using Vulkan layer settings through three different methods to match specific Vulkan developers' workflows:

- Using the Vulkan API: `vkCreateInstance()` and `VK_EXT_layer_settings`.
- Using the `vk_layer_settings.txt` file, that can be generated by the GUI interface called *Vulkan Configurator*.
- Using environment variables.

These three methods are implemented by the *Vulkan Layer Settings library* part of the *Vulkan-Utility-Libraries* repository. Any layer project that uses this library will provide these three methods to control layer settings, bringing consistency and ease of use of layers to the Vulkan community.

The *Vulkan Layer Settings library* is currently used by the *Vulkan Validation layer*, the *Vulkan Profiles layer*, the *Vulkan Extension layers* and the *LunarG Utility layers*.

Configuring layers means multiple tasks: Enabling layers; Ordering layers; Configuring the layers capabilities. These three aspects are described with each method to configure layers.



Example of system configured with enabled and ordered layers on the Vulkan developer system

Since a setting can be set via multiple methods simultaneously here is the priority order:

1. Environment variables (Which overrides the values set by the two others methods)
2. `vk_layer_settings.txt`
3. `VK_EXT_layer_settings` Vulkan extension

All the settings are described in the JSON layer manifest file that ships with the layer binary. When the settings are implemented in a layer using the Vulkan Layer Settings library, all the settings can be configured with all three methods.

Guideline: Settings which are unknown by the layer will be ignored independently of the method. It's the responsibility of the layer developer to ensure backward compatibility with previous versions of the layer. This is to ensure the list of layer settings remain relatively stable across versions and that the responsibility of handling layer backward compatibility doesn't fall on Vulkan application developers as this could quickly become untrackable.

Configuring Layers using the Vulkan API

Enabling and ordering the layer using `vkCreateInstance()`

Applications may programmatically activate layers via the `vkCreateInstance()` entry point. This is done by setting `enabledLayerCount` and `ppEnabledLayerNames` in the `VkInstanceCreateInfo` structure.

The layer names order in `ppEnabledLayerNames` specifies the layers execution ordering from closer to the Vulkan application to closer to the Vulkan driver.

Code example to enable and order the validation and the profiles layers programmatically:

```
C/C++
const VkApplicationInfo app_info = initAppInfo();

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

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

VkInstance instance = VK_NULL_HANDLE;
VkResult result = vkCreateInstance(&inst_create_info, nullptr, &instance);
```

In this example, the Khronos validation layer will be called *before* the Khronos profiles layer, it's called closer to the Vulkan application than the Vulkan driver.

Configuring the layer settings using VK_EXT_layer_settings

Layer settings may be configured using the `VK_EXT_layer_settings` extension by initializing the `VkLayerSettingsCreateInfoEXT` structure and chaining it to the `pNext` of `VkInstanceCreateInfo` when creating a Vulkan instance.

Code example to configure the validation layer programmatically:

```C/C++
const char* name = "VK_LAYER_KHRONOS_validation";

const VkBool32 setting_validate_core = VK_TRUE;
const VkBool32 setting_validate_sync = VK_TRUE;
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 VkBool32 setting_enable_message_limit = VK_TRUE;
const int32_t setting_duplicate_message_limit = 3;

const VkLayerSettingEXT settings[] = {
    {name, "validate_core", VK_LAYER_SETTING_TYPE_BOOL32_EXT,
     1, &setting_validate_core},
    {name, "validate_sync", VK_LAYER_SETTING_TYPE_BOOL32_EXT,
     1, &setting_validate_sync},
    {name, "thread_safety", VK_LAYER_SETTING_TYPE_BOOL32_EXT,
     1, &setting_thread_safety},
    {name, "debug_action", VK_LAYER_SETTING_TYPE_STRING_EXT,
     1, setting_debug_action},
    {name, "report_flags", VK_LAYER_SETTING_TYPE_STRING_EXT,
     static_cast<uint32_t>(std::size(setting_report_flags)),
     setting_report_flags}
    {name, "enable_message_limit", VK_LAYER_SETTING_TYPE_BOOL32_EXT,
     1, &setting_enable_message_limit},
    {name, "duplicate_message_limit", VK_LAYER_SETTING_TYPE_INT32_EXT,
     1, &setting_duplicate_message_limit}};

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[] = {name};
const char* extensions[] = {VK_EXT_LAYER_SETTINGS_EXTENSION_NAME};
```

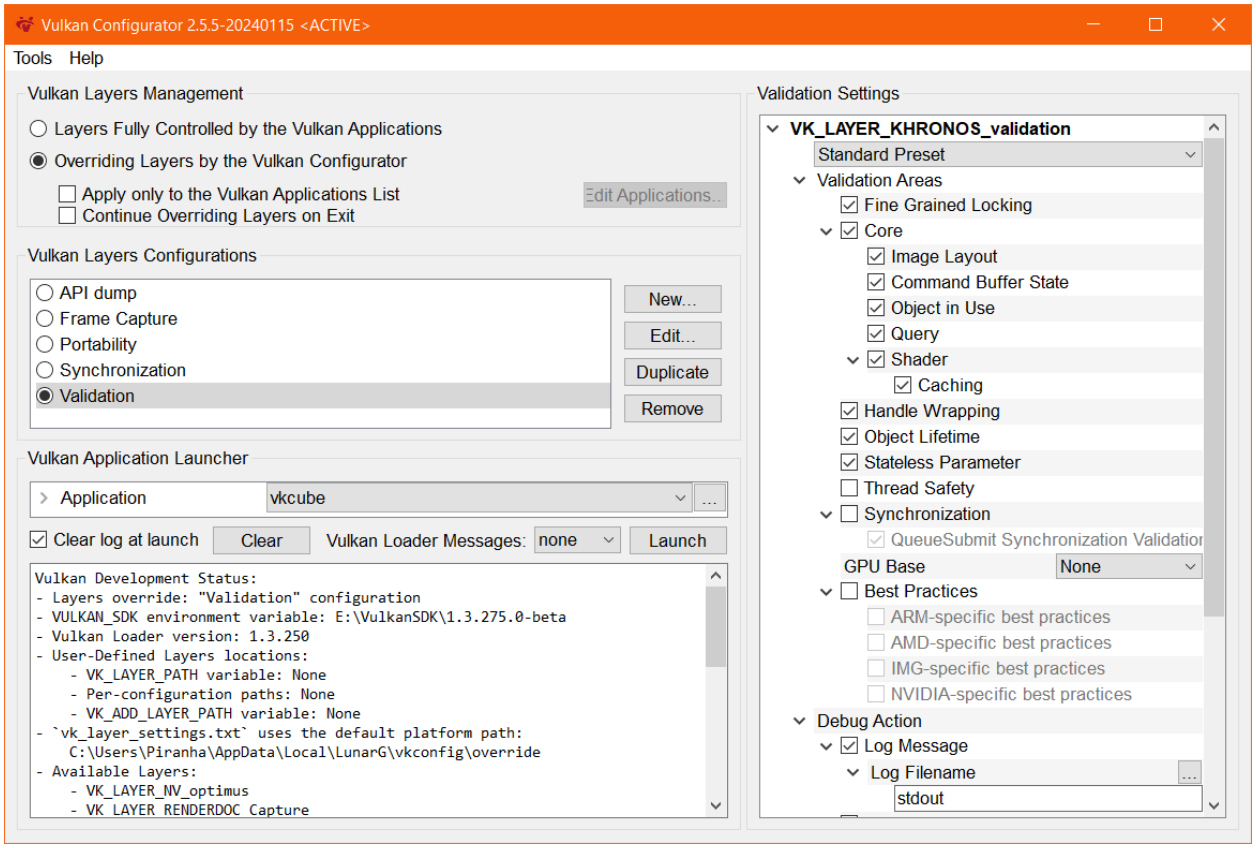
```
const VkInstanceCreateInfo inst_create_info = {
    VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO, &layer_settings_create_info,
    0,
    &app_info,
    static_cast<uint32_t>(std::size(layers)), layers,
    static_cast<uint32_t>(std::size(extensions)), extensions
};

VkInstance instance = VK_NULL_HANDLE;

VkResult result = vkCreateInstance(
    &inst_create_info, nullptr, &instance);
```

Configuring Layers using Vulkan Configurator

Vulkan developers can configure layers through a graphical user interface. *Vulkan Configurator* allows full user control of Vulkan layers, including enabling or disabling specific layers, controlling layer order, changing layer settings, etc. *Vulkan Configurator* configures the layers by applying a global system configuration of the Vulkan loader and creating a `vk_layer_settings.txt` file that will be found by any layer.

Vulkan Configurator can be used using the command line to configure the system environment. Use the command `vkconfig --help` for more information.

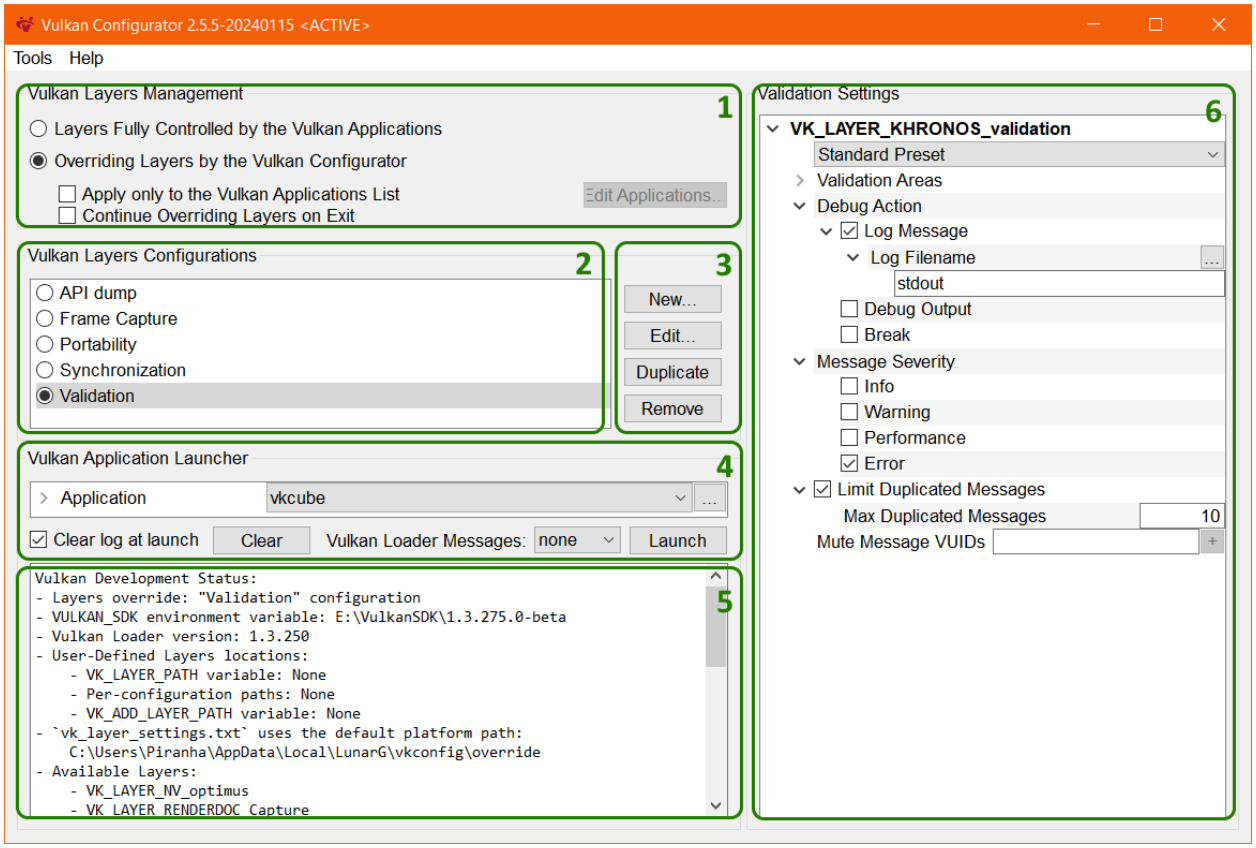
We recommend using the *Vulkan Configurator* GUI approach for Vulkan application developers. It's the most effective approach to switch between multiple layer configurations and quickly iterate during development. Additionally, *Vulkan Configurator* presents to the Vulkan application developers the layers found on the system and the settings of each layer, allowing Vulkan application developers to discover new functionality from the GUI without having to dig into each layer's documentation.



The *Vulkan Configurator* interface

Before *Vulkan Configurator*, a Vulkan developer would have to configure the layers either programmatically or by using environment variables specified by the layers documentation, which required a significant and continuous learning curve as the Vulkan layers capabilities evolved.

Vulkan Configurator was created to present the Vulkan layers with an intuitive interface enabling developers to use layer features with existing Vulkan applications, instantly and dramatically reducing development iteration time as no compilation, no learning of the new settings, and no tracking of the new features is required. The features are directly available in the GUI.



The Vulkan Configurator UI comprises six areas:

1) Vulkan Layers Management: this area controls whether the Vulkan Layers override is active or not. It also determines whether the override is applied only to a selection of Vulkan applications or to all Vulkan applications. Finally, this area

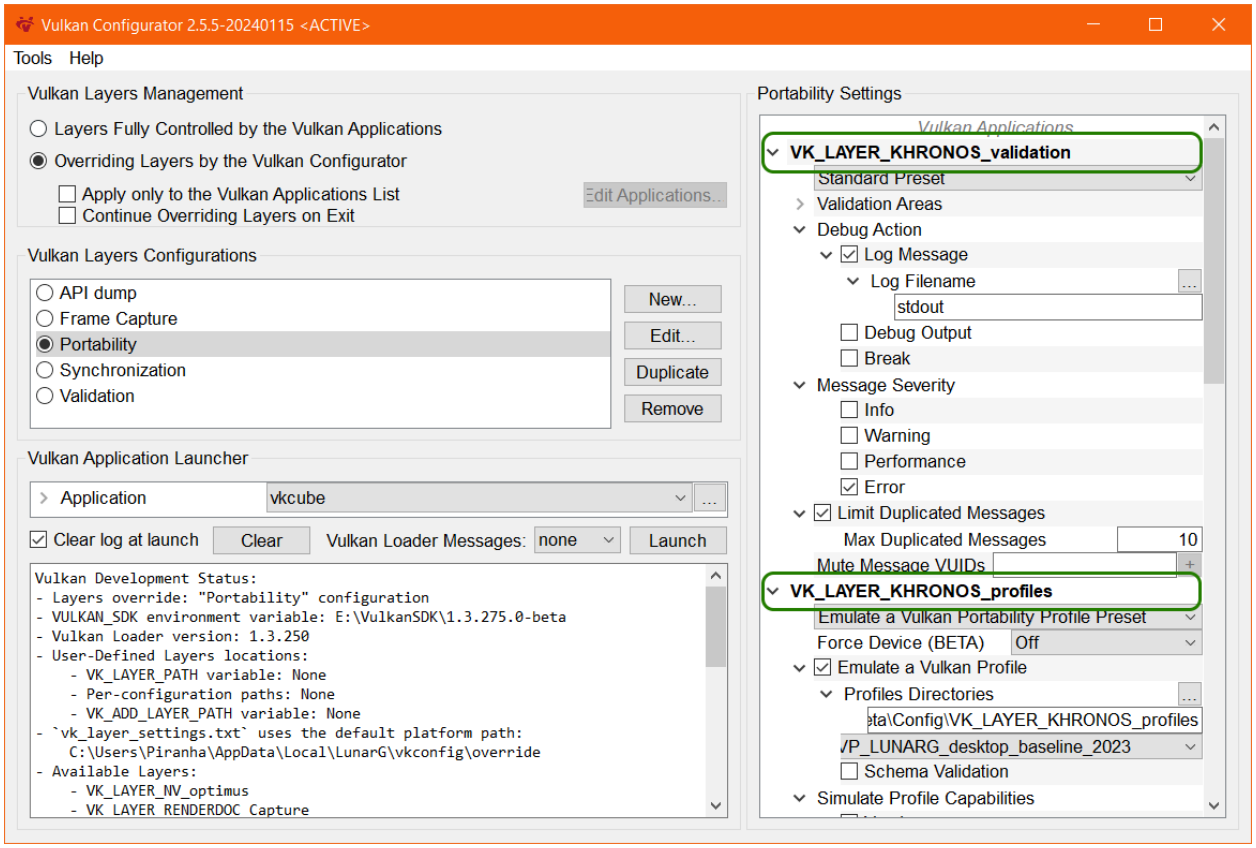
specifies whether the override remains active or not when Vulkan Configurator is closed.

2) Vulkan Layers Configurations: the list of pre-configured layers configurations. Vulkan Configurator is installed with a selection of built-in configurations that are listed on the screenshot. Each built-in configuration is designed to handle a specific Vulkan application developer use case. Using the context menu, we can design user-defined layer configurations to create layers configurations for our specific use cases.
3) Create a new layers configuration ; edit, duplicate or remove the selected layers configuration. The “Edit…” button allows opening the “*Edit Vulkan Layers*” window to select the layers behavior with the following actions:
    a) to override,
    b) to exclude,
    c) or to be handled by the Vulkan applications.

    The “*Edit Vulkan Layers...*” window allows adding paths to find additional layers on the system.

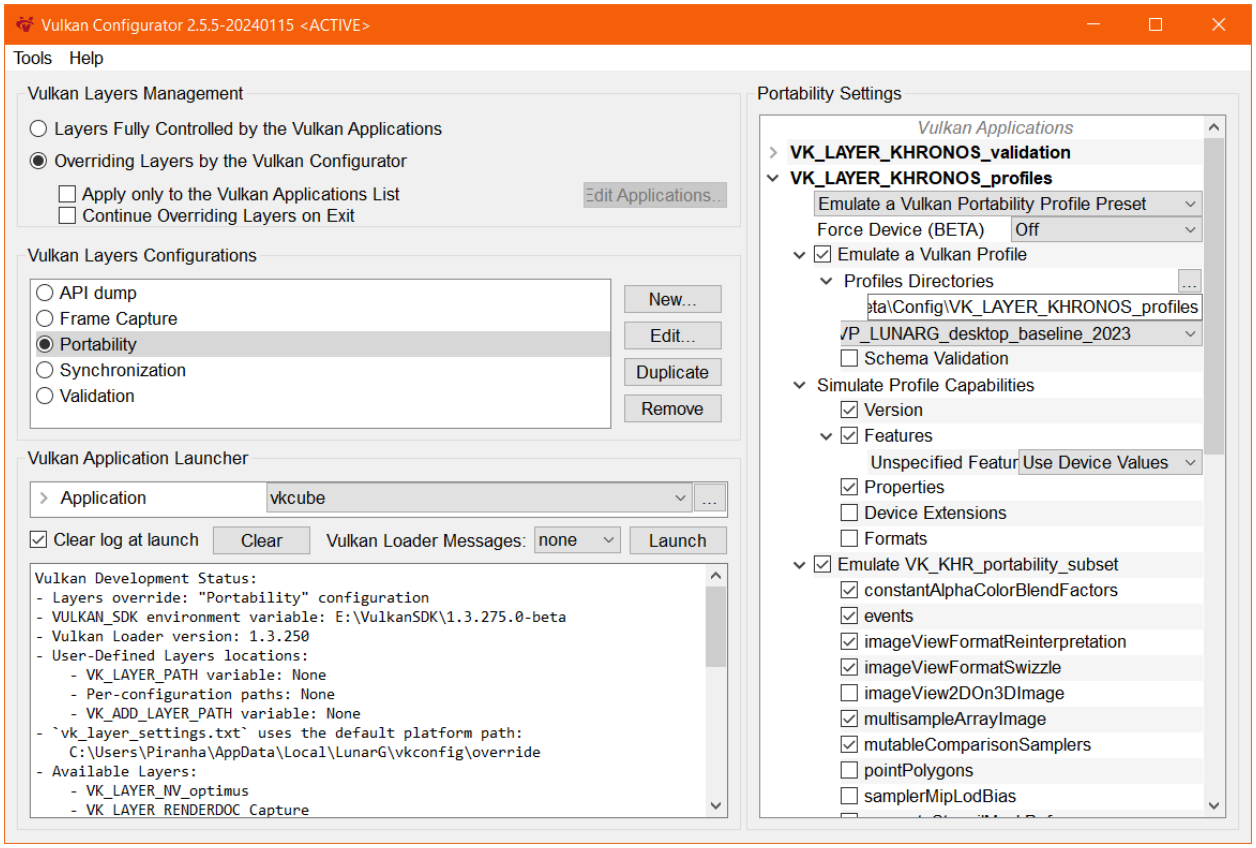
4) Vulkan Application Launcher: this area allows running any Vulkan application with the selected layers configuration.
5) Log window: on start-up, when selecting a layer configuration or updating the layers list of a layer configuration, the log window will display the “Vulkan Development Status” which reports the version of various components, relevant paths for Vulkan developers, and the list of available layers. When launching a Vulkan application from Vulkan Configurator, the log window will display anything sent to stdout or stderr from the Vulkan layers, Vulkan applications, and the Vulkan Loader.
6) Layers configuration settings: the tree of settings for each layer. If the layers have setting presets, they are displayed just below the layer name.

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

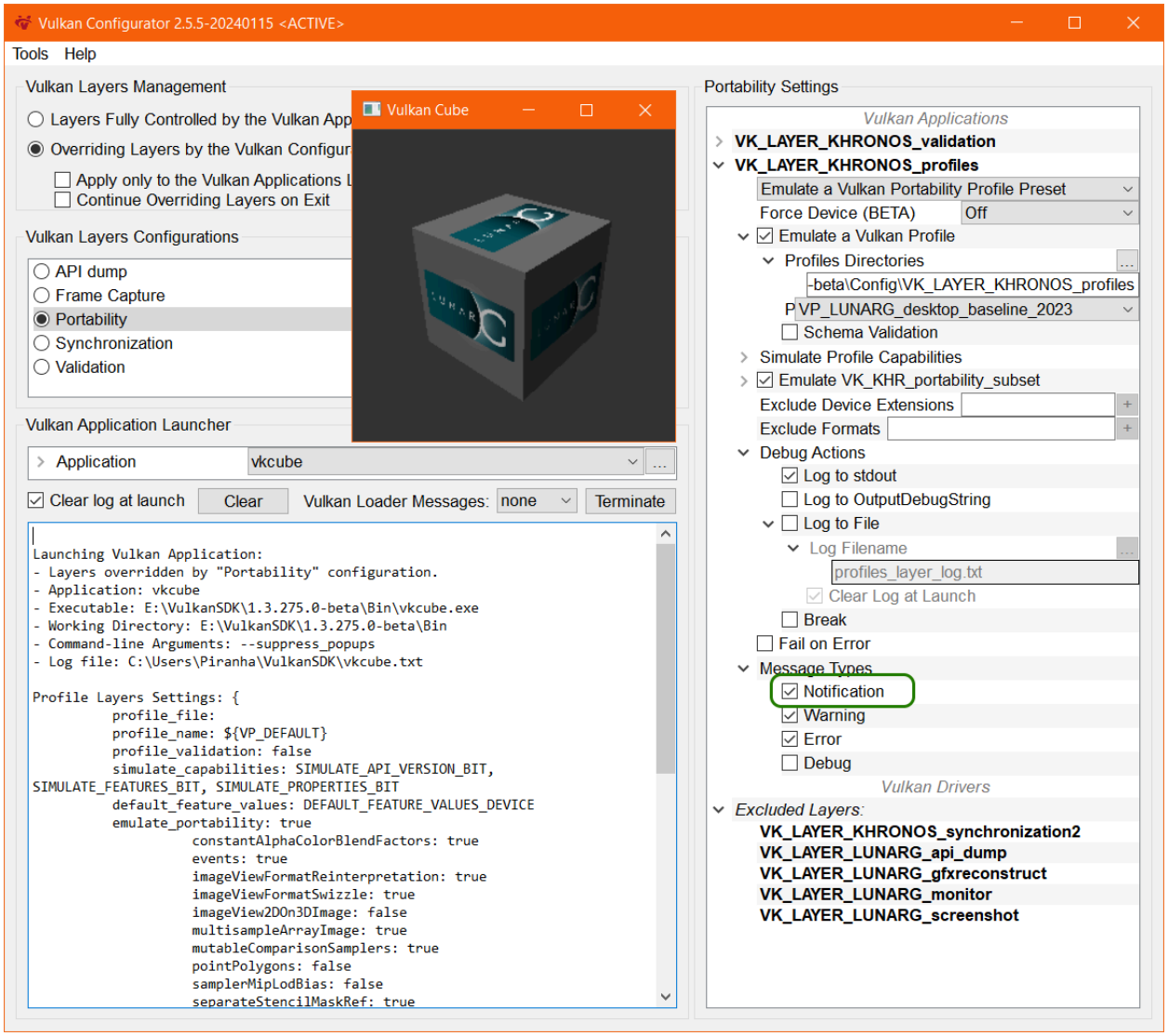


This configuration includes the *Vulkan Validation layer* and the *Vulkan Profiles layer*.

On the right panel, we can see the layers settings.



We can hide the *layer* settings of a specific layer by clicking the carrot next to the layer name.



For more information about the tool is available in the [Vulkan Configurator documentation]

Enabling and ordering layers (VkLayer_override.json)

To control the enabled layers and the layer order, *Vulkan Configurator* generates the `VkLayer_override.json` file, which is consumed by the Vulkan loader to enable Vulkan layers and control the order of the layers. This file also stores the user-defined paths specified in *Vulkan Configurator* to find additional layers.

The override layer file on Linux and macOS

Unix systems store override layer file in the following paths:

- `$HOME/.local/share/vulkan/implicit_layer.d/VkLayer_override.json`

The override layer file on Windows

Windows systems store the override layer file in the following path:

- `%HOME%\AppData\Local\LunarG\vkconfig\override\VkLayerOverride.json`

Configuring the layers (vk_layer_settings.txt)

To control the layer settings, *Vulkan Configurator* generates the `vk_layer_settings.txt` file which is consumed by the Vulkan layers and sets the setting values defined by the Vulkan developers using the UI.

By default, the Vulkan Layer Settings library requires the settings file to be named `vk_layer_settings.txt` and it will search it in the working directory of the targeted application. Hence, if a file is found in the working directory of the targeted application, the Vulkan Layer Settings library will bypass the layer settings created by *Vulkan Configurator*. If `VK_LAYER_SETTINGS_PATH` is set and is a directory, then the settings file must be a file called `vk_layer_settings.txt` in the directory given by `VK_LAYER_SETTINGS_PATH`. If `VK_LAYER_SETTINGS_PATH` is set and is not a directory, then it must point to a file (with any name) which is the layer settings file.

The settings file can be created, modified or generated by the Vulkan application developers or third party tools. The settings file consists of comment lines and settings lines. Comment lines begin with the `#` character. Settings lines have the following format:

<LayerName>.<setting_name> = <setting_value>

The list of available settings is available in the layer manifest.

Example of `vk_layer_settings.txt` file:

```
Unset
# The main, heavy-duty validation checks. This may be valuable early in the
# development cycle to reduce validation output while correcting
# parameter/object usage errors.
khronos_validation.validate_core = true

# Enable synchronization validation during command buffers recording. This
# feature reports resource access conflicts due to missing or incorrect
```

```
# synchronization operations between actions (Draw, Copy, Dispatch, Blit)
# reading or writing the same regions of memory.
khronos_validation.validate_sync = true

# Thread checks. In order to not degrade performance, it might be best to run
# your program with thread-checking disabled most of the time, enabling it
# occasionally for a quick sanity check or when debugging difficult
# application behaviors.
khronos_validation.thread_safety = true

# Specifies what action is to be taken when a layer reports information
khronos_validation.debug_action = VK_DBG_LAYER_ACTION_LOG_MSG

# Comma-delineated list of options specifying the types of messages to be
# reported
khronos_validation.report_flags = debug,error,perf,info,warn

# Enable limiting of duplicate messages.
khronos_validation.enable_message_limit = true

# Maximum number of times any single validation message should be reported.
khronos_validation.duplicate_message_limit = 3
```

Layer Settings File location on Linux and macOS

Unix systems store the layer setting file in the following path:

- `$HOME/.local/share/vulkan/settings.d/vk_layer_settings.txt`

Layer Settings File location on Windows

Windows systems store the layer setting file in the following path:

- `%HOME%\AppData\Local\LunarG\vkconfig\override\vk_layer_settings.txt`

Configuring Layers using Environment Variables

Finding Vulkan Layers

In order to enable a Vulkan layer from the command-line, you must first make sure:

1. The layer's Manifest JSON file is found by the Vulkan Desktop Loader because it is in:
    - One of the standard operating system install paths
    - It was added using one of the layer path environment variables (`VK_LAYER_PATH` or `VK_ADD_LAYER_PATH`).
    - See the `Layer Discovery` section of the Vulkan Loader's Layer Interface doc.
2. The layer's library file is able to be loaded by the Vulkan Desktop Loader because it is in:
    - A standard library path for the operating system
    - The library path has been updated using an operating system-specific mechanism such as:
        - Linux: adding the path to the layer's library .so with `LD_LIBRARY_PATH`
        - MacOS: adding the path to the layer's library .dylib with `DYLD_LIBRARY_PATH`
3. The layer's library file is compiled for the same target and bitdepth (32 vs 64) as the application

Activating Specific SDK Layers

To activate layers located in a particular SDK installation, or layers built locally from source, specify the layer JSON manifest file directory using either `VK_LAYER_PATH` or `VK_ADD_LAYER_PATH`. The difference between `VK_LAYER_PATH` and `VK_ADD_LAYER_PATH` is that `VK_LAYER_PATH` overrides the system layer paths so that no system layers are loaded by default unless their path is added to the environment variable. `VK_ADD_LAYER_PATH` on the other hand, causes the loader to search the additional layer paths listed in the environment variable first, and then the standard system paths will be searched.

Usages on each desktop platform

For example, if a Vulkan SDK is installed in `C:\VulkanSDK\1.3.261.0`, execute the following in a Command Window:

```
Unset
C:\> set VK_LAYER_PATH=C:\VulkanSDK\1.3.261.0\Bin
```

For Linux, if Vulkan SDK 1.3.261.0 was locally installed in `/sdk` and `VULKAN_SDK=/sdk/1.3.261.0/x86_64`:

```
Unset
$ export VK_LAYER_PATH=$VULKAN_SDK/lib/vulkan/layers
$ export LD_LIBRARY_PATH=$VULKAN_SDK/lib:$VULKAN_SDK/lib/vulkan/layers
```

For macOS, if Vulkan SDK 1.3.261.0 was locally installed in `/sdk` and `VULKAN_SDK=/sdk/1.3.261/macOS`:

```
Unset
$ export VK_LAYER_PATH=$VULKAN_SDK/share/vulkan/explicit_layers.d
$ export DYLD_LIBRARY_PATH=$VULKAN_SDK/lib
```

Enabling and ordering Vulkan Layers

Originally, the Vulkan Desktop Loader provided `VK_INSTANCE_LAYERS` to enable layers from the command-line. However, starting with the Vulkan Loader built against the 1.3.234 Vulkan headers, the `VK_LOADER_LAYERS_ENABLE` environment variable was added to allow for more easily enabling Vulkan layers. The newer Loaders will continue to accept the original `VK_INSTANCE_LAYERS` environment variable for some time, but it is considered deprecated.

Vulkan 1.3.234 Loader and Newer (VK_LOADER_LAYERS_ENABLE)

The easiest way to enable a layer with a more recent drop of the Vulkan Loader is using the `VK_LOADER_LAYERS_ENABLE` environment variable. This environment variable accepts a case-insensitive, comma-delimited list of globs which can be used to define the layers to load.

For example, previously if you wanted to enable the Profiles layer and the Validation layer, you would have to set `VK_INSTANCE_LAYERS` equal to the full name of each layer:

```
Unset
VK_INSTANCE_LAYERS=VK_LAYER_KHRONOS_validation;VK_LAYER_KHRONOS_profiles
```

Now, with `VK_LOADER_LAYERS_ENABLE`, you simply can use stars where you don't want to fill in the full name:

```
Unset
VK_LOADER_LAYERS_ENABLE=*validation,*profiles
```

Note that order is relevant, with the initial layer being the closest to the application, and the final layer being closest to the driver. In this example, the Khronos validation layer will be called *before* the Khronos profiles layer.

Usages on each desktop platform

Example Usage On Windows:

```
Unset
C:\> set VK_LOADER_LAYERS_ENABLE=*validation,*profiles
```

Example Usage On Linux/macOS:

```
Unset
$ export VK_LOADER_LAYERS_ENABLE=*validation,*profiles
```

More info about the new layer filtering environment variables can be found in the `Layer Filtering` section of the Loader Layer Documentation.

Older Vulkan Loaders (VK_INSTANCE_LAYERS)

Vulkan Desktop loaders version 1.3.233 and below will not accept the filtering environment variable, and so must continue using the original `VK_INSTANCE_LAYERS` environment variable.

Example Usage On Windows:

The variable should include a semicolon-separated list of layer names to activate. Note that order is relevant, with the initial layer being the closest to the application, and the final layer being closest to the driver.

```
Unset
C:\> set
VK_INSTANCE_LAYERS=VK_LAYER_KHRONOS_validation;VK_LAYER_KHRONOS_profiles
```

In this example, the Khronos validation layer will be called *before* the Khronos profiles layer. `VK_INSTANCE_LAYERS` may also be set in the system environment variables.

Example Usage On Linux/macOS:

The variable should include a colon-separated list of layer names to activate. Note that order is relevant, with the initial layer being the closest to the application, and the final layer being closest to the driver.

```
Unset
$ export
VK_INSTANCE_LAYERS=VK_LAYER_KHRONOS_validation:VK_LAYER_KHRONOS_profiles
```

In this example, the Khronos validation layer will be called *before* the Khronos profiles layer.

Layer Settings Environment Variables

The settings can also be set using environment variables. The settings that can be set using environment variables are listed in the documentation for each supported layer. If an environment variable is set, its value takes precedence over the value in the settings file.

The environment variable names for the layer settings have multiple variants that follows the format:

- VK_<*LayerVendor*>_<*LayerName**><**setting_name***> *which take precedence over:*
- *VK_<*LayerName*><*setting_name*> which take precedence over:*

- VK_<*setting_name*>`

This approach allows sharing the same setting name for potentially multiple layers but still use different values for the same setting name if this is what is required for the Vulkan developer use case.

Examples of environment variable variants for a single setting:

- VK_KHRONOS_VALIDATION_DEBUG_ACTION
- VK_VALIDATION_DEBUG_ACTION
- VK_DEBUG_ACTION

Examples Usage on Windows:

```
Unset
C:\> set VK_VALIDATION_VALIDATE_CORE=true
C:\> set VK_VALIDATION_VALIDATE_SYNC=true
C:\> set VK_VALIDATION_THREAD_SAFETY=true
C:\> set VK_VALIDATION_DEBUG_ACTION=VK_DBG_LAYER_ACTION_LOG_MSG
C:\> set VK_VALIDATION_REPORT_FLAGS=debug;error;perf;info;warn
C:\> set VK_VALIDATION_ENABLE_MESSAGE_LIMIT=true
C:\> set VK_VALIDATION_DUPLICATE_MESSAGE_LIMIT=3
```

Examples Usage on Linux/macOS:

```
Unset
$ export VK_VALIDATION_VALIDATE_CORE=true
$ export VK_VALIDATION_VALIDATE_SYNC=true
$ export VK_VALIDATION_THREAD_SAFETY=true
$ export VK_VALIDATION_DEBUG_ACTION=VK_DBG_LAYER_ACTION_LOG_MSG
$ export VK_VALIDATION_REPORT_FLAGS=debug:error:perf:info:warn
$ export VK_VALIDATION_ENABLE_MESSAGE_LIMIT=true
$ export VK_VALIDATION_DUPLICATE_MESSAGE_LIMIT=3
```

Revision History

Revision Date	SDK Release	Comments
April 2024	SDK 1.3.280.0	- Fix VK_EXT_layer_settings usage example.
January 2024	SDK 1.3.275.0	- Initial release.