

Configuring Vulkan Layers



Christophe Riccio, [LunarG](#)
May 2026

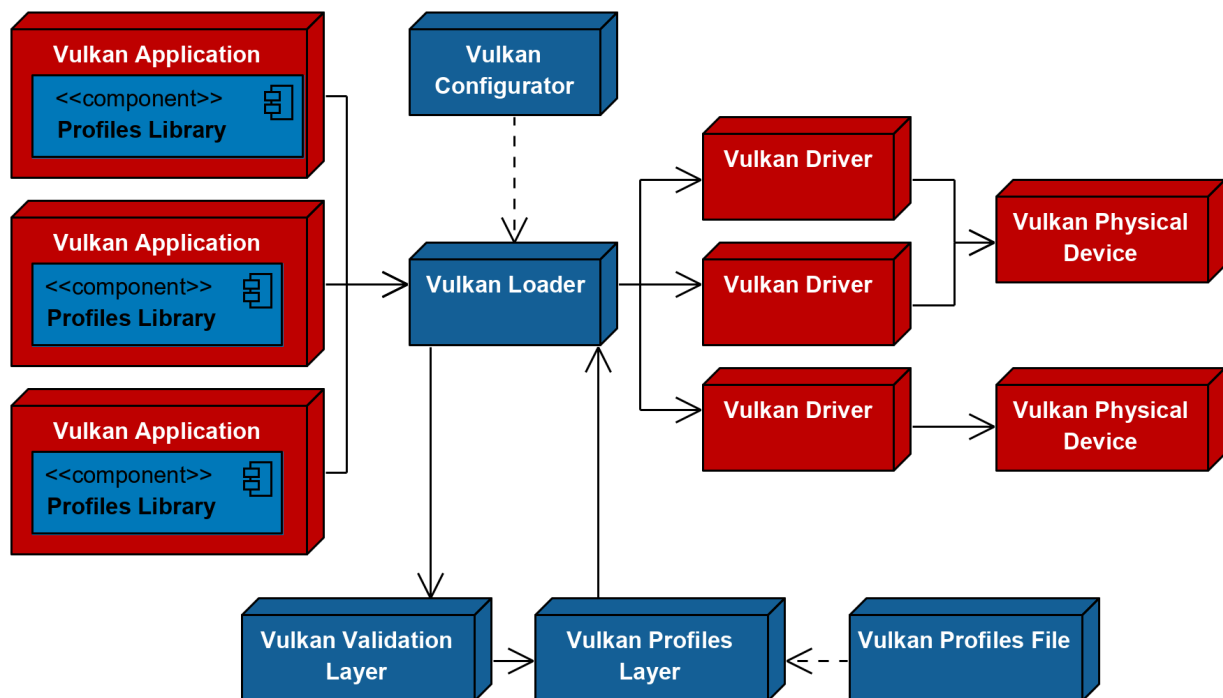
Configuring Vulkan Layers approaches	4
Three approaches	4
Backward Compatibility Guideline	5
Deprecation Notice	5
Configuring Layers using the Vulkan API	6
Enabling and ordering the layer using vkCreateInstance()	6
Initializing layer settings using VK_EXT_layer_settings	6
Configuring Layers using Vulkan system files	9
The Vulkan Configurator interface	10
Vulkan Configurator tabs description	10
Enabling and ordering layers using vk_loader_settings.json	12
Initializing Layer Settings using vk_layer_settings.txt	12
vk_layer_settings.txt location	12
vk_layer_settings.txt syntax	13
Configuring Layers using Environment Variables	15
Finding Vulkan Layers	15
Usages on each desktop platform	15
Enabling and ordering layers with VK_INSTANCE_LAYERS	16
Vulkan 1.3.234 Loader and Newer (VK_LOADER_LAYERS_ENABLE)	16
Usages on each desktop platform	17
Older Vulkan Loaders (VK_INSTANCE_LAYERS)	17
Initializing Layer Settings with Environment Variables	18
Examples	19
Generating Vulkan layers settings files	20
Generating the layer settings documentation	20
Context menu to generate the Validation layers configuration files	20
Generating layer settings code for each layer settings approach	20
Using VK_EXT_layer_settings to configure layers programmatically	21
Generated C++ helper library example subset	21
Generated C++ helper library usage example	22
Using environment variables to configure layers	22
Generated Environment Variables script example subset	22
Using command line to generate the layer settings files	23
Vulkan Configurator command line to generate layer settings files	25
Vulkan SDK generated layer settings files	25
vk_layer_settings.txt	25
vk_layer_settings.sh	26

vk_layer_settings.bat	26
vulkan_layer_settings.hpp	26
Revision History	27

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.

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.



Example: A system configured with enabled and ordered layers on a Vulkan developer's system

Three approaches

A layers configuration consists in two operations:

- Selecting and ordering layers for the Vulkan Loader.
- Configuring each layer themselves using layer settings.

Vulkan layers can be configured using three different methods that match different Vulkan development workflows:

- **Using environment variables:** [Loader environment variables](#) and [per-layer settings environment variables](#).

- **Using dedicated Vulkan system files:** [vk_loader_settings.json](#) and [vk_layer_settings.txt](#).
- **Using the Vulkan API, programmatically in the Vulkan application:** [vkCreateInstance](#) and [VK_EXT_layer_settings](#) extension.

When a setting can be set via multiple methods simultaneously, the priority order is:

1. Using environment variables (highest priority).
2. Using dedicated Vulkan system files.
3. Using the Vulkan API, programmatically in the Vulkan application (lowest priority).

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](#).

Each setting is 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.

Backward Compatibility Guideline

Settings unknown to the layer are ignored, regardless of method. Layer developers are responsible for ensuring backward compatibility with previous layer versions. This keeps the list of layer settings relatively stable across versions and avoids burdening Vulkan application developers with unmanageable compatibility tracking.

Deprecation Notice

This document describes the Vulkan Layers configuration method implemented by the Vulkan Loader 1.3.304 and later, relying on the `vk_loader_settings.json` file implemented by *Vulkan Configurator* 3. The previous approach using `VkLayer_override.json` is deprecated.

Configuring Layers using the Vulkan API

Enabling and ordering the layer using vkCreateInstance()

Applications can enable and order layers programmatically when calling `vkCreateInstance()`. This is done by setting `enabledLayerCount` and `ppEnabledLayerNames` in the `VkInstanceCreateInfo` structure.

The layer names order in `ppEnabledLayerNames` determines the execution order:

- First layer in the list → closest to the application (called first)
- Last layer in the list → closest to the driver

```
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);
```

Example: The Khronos Validation layer executes before the Khronos Profiles layer

Initializing layer settings using VK_EXT_layer_settings

Layer settings can be configured programmatically 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);
```

Example: Configuring Khronos Validation layer settings programmatically

Configuring Layers using Vulkan system files

Two system files can be created for persistent Vulkan layers configuration:

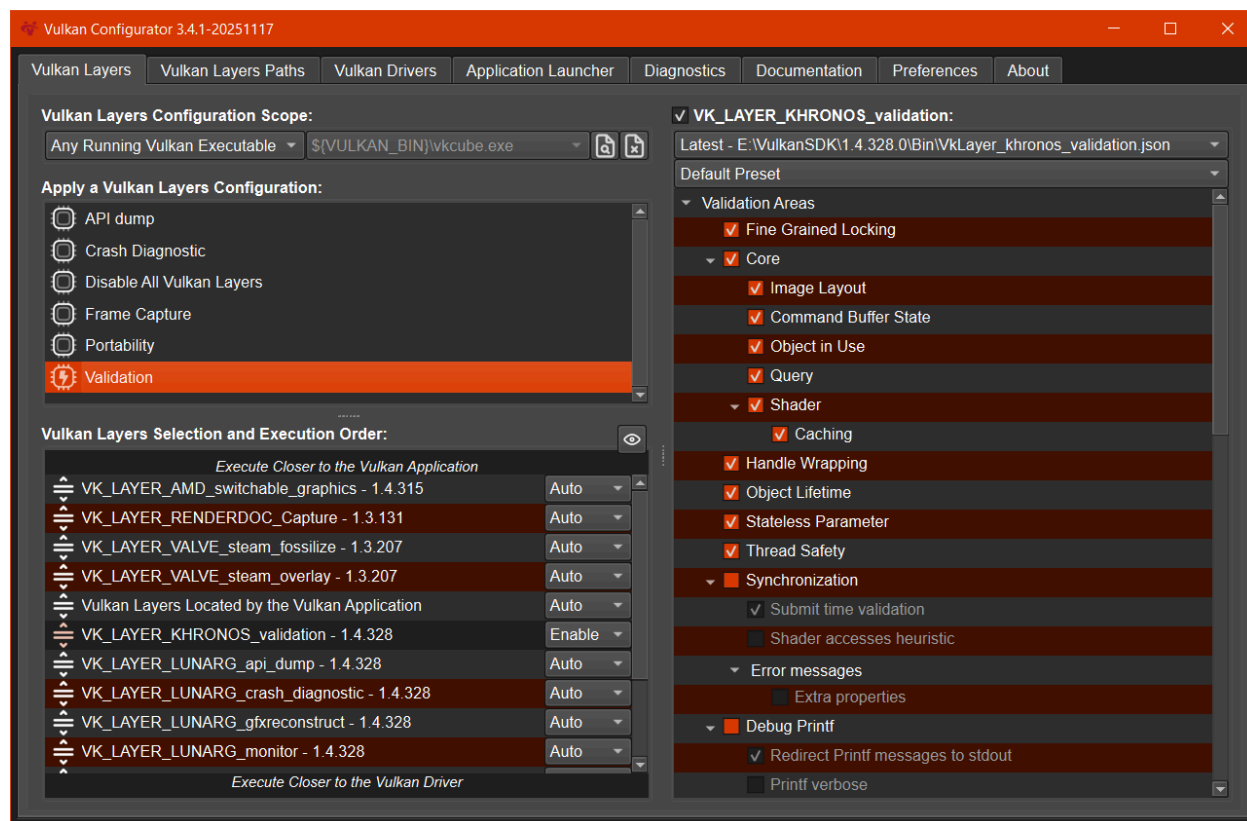
- `vk_loader_settings.json`: read by the Vulkan Loader to enable layers and set their execution order
- `vk_layer_settings.txt`: read by the layers themselves to apply individual settings

[Vulkan Configurator](#) uses this method to configure Vulkan Layers. It creates these files while running and cleans them up on exit unless configured differently in the *preferences* tab.

Vulkan Configurator also provides functionalities to:

- Generate layer settings files to control layers without Vulkan Configurator
- Select and prioritize physical devices
- Load additional Vulkan drivers
- Generate diagnostic logs

The tool also works from the command line (`vkconfig --help` for details).



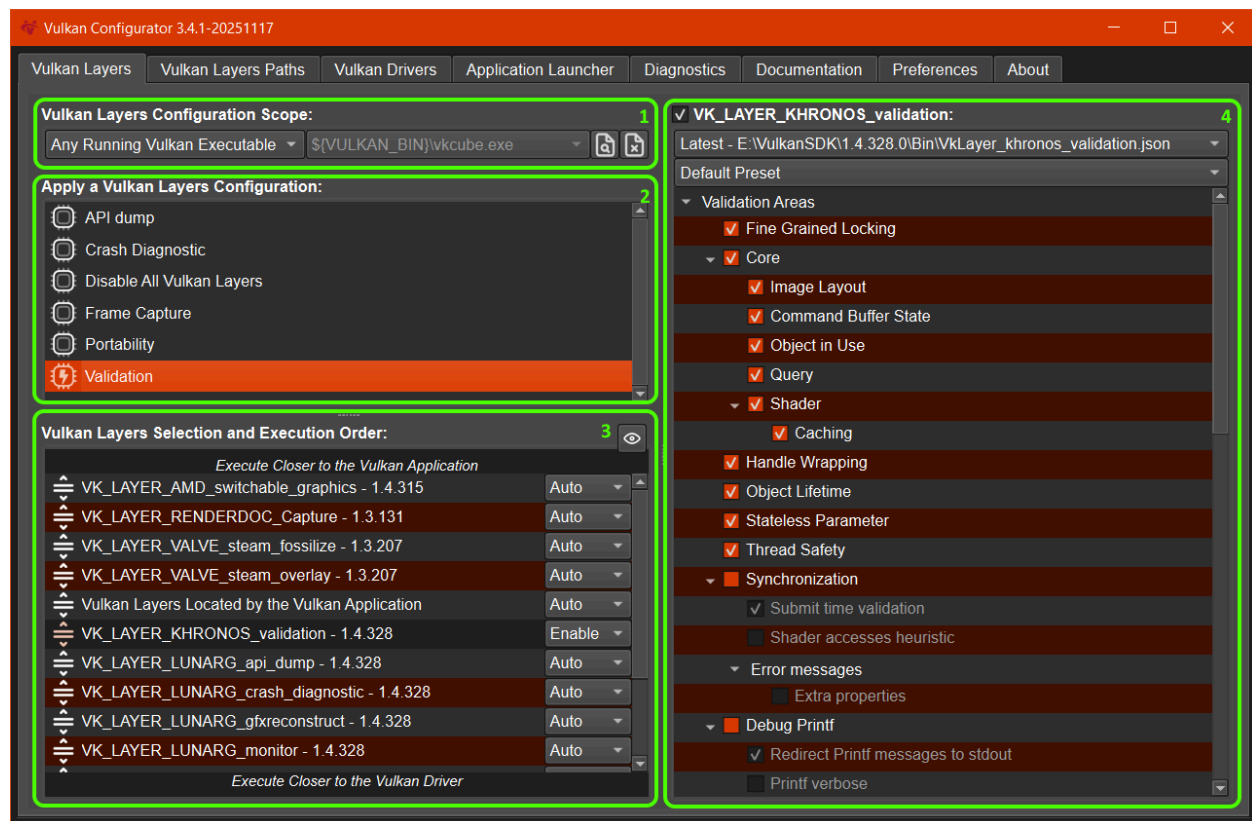
We recommend the Vulkan Configurator GUI for Vulkan application developers because it lets you quickly switch configurations, iterate during development, and discover available layers and settings without reading extensive documentation.

The *Vulkan Configurator* interface

Before *Vulkan Configurator* existed, developers had to configure layers either in code or via environment variables listed in each layer's documentation. This required a steep and ongoing learning curve as layers evolved.

Vulkan Configurator solves this issue by providing a visual and intuitive interface to preset the layers and settings for the layers configuration.

Vulkan Configurator tabs description



The Vulkan Configurator “Vulkan Layers” tab has 4 sections:

- 1) **Vulkan Layers Configuration Scope:** This section controls whether the Vulkan Layers configuration is applied on all Vulkan executables or a specific list of Vulkan executables.

- 2) **Apply a Vulkan Layers Configuration:** This section contains a list of saved layers configurations including built-in configurations for common use cases.

Using the context menu, we can:

- Create new configurations
- Generate environment variable scripts (.sh, .bat files)
- Generate C++ helper code for VK_EXT_layer_settings (.hpp file)
- Etc.

This code generation allows Vulkan application developers to create and test layer configurations using the user interface and export the configurations for [use in environment variables scenarios](#) or [use directly in the Vulkan application code](#).

- 3) **Vulkan Layers Selection and Execution Order:** The section shows all layers found on the system and their execution order.
- 4) **Vulkan Layer Settings:** This section profiles a tree view of all the settings for the selected layer. If the layer provides setting presets, they are displayed just below the layer name.

Enabling and ordering layers using vk_loader_settings.json

Introduced with Vulkan Configurator 3 and Vulkan Loader 1.4.304, the `vk_loader_settings.json` file:

- Controls which layers are enabled
- Sets layers execution order
- Stores custom layer search paths
- And select or order Vulkan physical devices.

The default location for the `vk_loader_settings.json` file is:

- On Linux and macOS:
`$HOME/.local/share/vulkan/loader_settings.d/vk_loader_settings.json`
- Windows:
`%HOME%\AppData\Local\LunarG\vulkan\vk_loader_settings.json`

Initializing Layer Settings using vk_layer_settings.txt

To initialize the layer settings, *Vulkan Configurator* generates the `vk_layer_settings.txt` file. This file is read by the layer during `vkCreateInstance` execution in the Vulkan application.

vk_layer_settings.txt location

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. If a `vk_layer_settings.txt` file is found in the working directory of the Vulkan application, the `vk_layer_settings.txt` file stored at the global system location is ignored.

The default global system location for the `vk_layer_settings.txt` file is:

- On Linux and macOS:
`$HOME/.local/share/vulkan/settings.d/vk_layer_settings.txt`
- On Windows:
`%HOME%\AppData\Local\LunarG\vkconfig\override\vk_layer_settings.txt`

On Windows, the registry store the default location at the following entry:

- `HKEY_CURRENT_USER\Software\Khronos\Vulkan\Settings`

The global system location of the layer settings can be overridden using the `VK_LAYER_SETTINGS_PATH` environment variable:

- If `VK_LAYER_SETTINGS_PATH` is set to a directory, then the settings file must be a file called `vk_layer_settings.txt`.
- If `VK_LAYER_SETTINGS_PATH` is set to a full path, the layer settings file can have any filename.

vk_layer_settings.txt syntax

The settings file can be created, modified or generated by the Vulkan application developers or third party tools.

The settings file consists of lines which can be comments or setting initializations. 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.

```
None
# 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
```

Example: vk_layer_settings.txt file for use with the validation layer

Configuring Layers using Environment Variables

Locating Vulkan Layers

To enable a Vulkan layer from the command-line, the Vulkan Loader must be able to locate and load it:

1. The layer's Manifest JSON file is found by the Vulkan 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 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

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.

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

```
None
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:

None

```
$ 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`:

None

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

Enabling and ordering layers with VK_INSTANCE_LAYERS

`VK_INSTANCE_LAYERS` is the environment variable used to enable and order layers when using a console.

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

On Windows (Command Prompt), the variable should include a **semicolon-separated** list of layer names to activate:

None

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

On Linux/macOS (shell), the variable should include a **colon-separated** list of layer names to activate:

None

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

Enabling layers with **VK_LOADER_LAYERS_ENABLE**

Vulkan Loader version 1.3.234 introduced the `VK_LOADER_LAYERS_ENABLE` environment variable. It accepts a case-insensitive, comma-delimited list of globs which can be used to define the layers to load.

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

None

```
VK_INSTANCE_LAYERS=VK_LAYER_KHRONOS_validation;VK_LAYER_KHRONOS_profiles
```

With `VK_LOADER_LAYERS_ENABLE`, we simply can use stars where we don't want to fill in the full name:

None

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

Usage on Windows

None

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

Usage On Linux/macOS

More info about the new layer filtering environment variables can be found in the Layer Filtering section of the [Loader Layer Documentation](#).

Warnings: `VK_LOADER_LAYERS_ENABLE` has limitations when interacting with the other methods to enable and order layers (Using `vk_loader_settings.json` or the Vulkan API), hence it is not recommended for such use cases.

Initializing Layer Settings with Environment Variables

Each layer setting can be initialized using environment variables.

If an environment variable is set, it overrides the corresponding value set in the `vk_layer_settings.txt` file or in the Vulkan application code.

There are multiple environment variable names for each layer setting. The naming convention is the following format:

- `VK_<LayerVendor>_<*LayerName*><*setting_name*>` (highest priority)
- `VK_<*LayerName*><*setting_name*>`
- `VK_<*setting_name*>` (lowest priority).

This allows the same setting name to be shared across layers while still permitting different values when needed.

Example of environment variable variants for a single setting:

- `VK_KHRONOS_VALIDATION_DEBUG_ACTION`
- `VK_VALIDATION_DEBUG_ACTION`
- `VK_DEBUG_ACTION`

None

```
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
```

Usage on Windows

None

```
$ 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
```

Usage on Linux/macOS

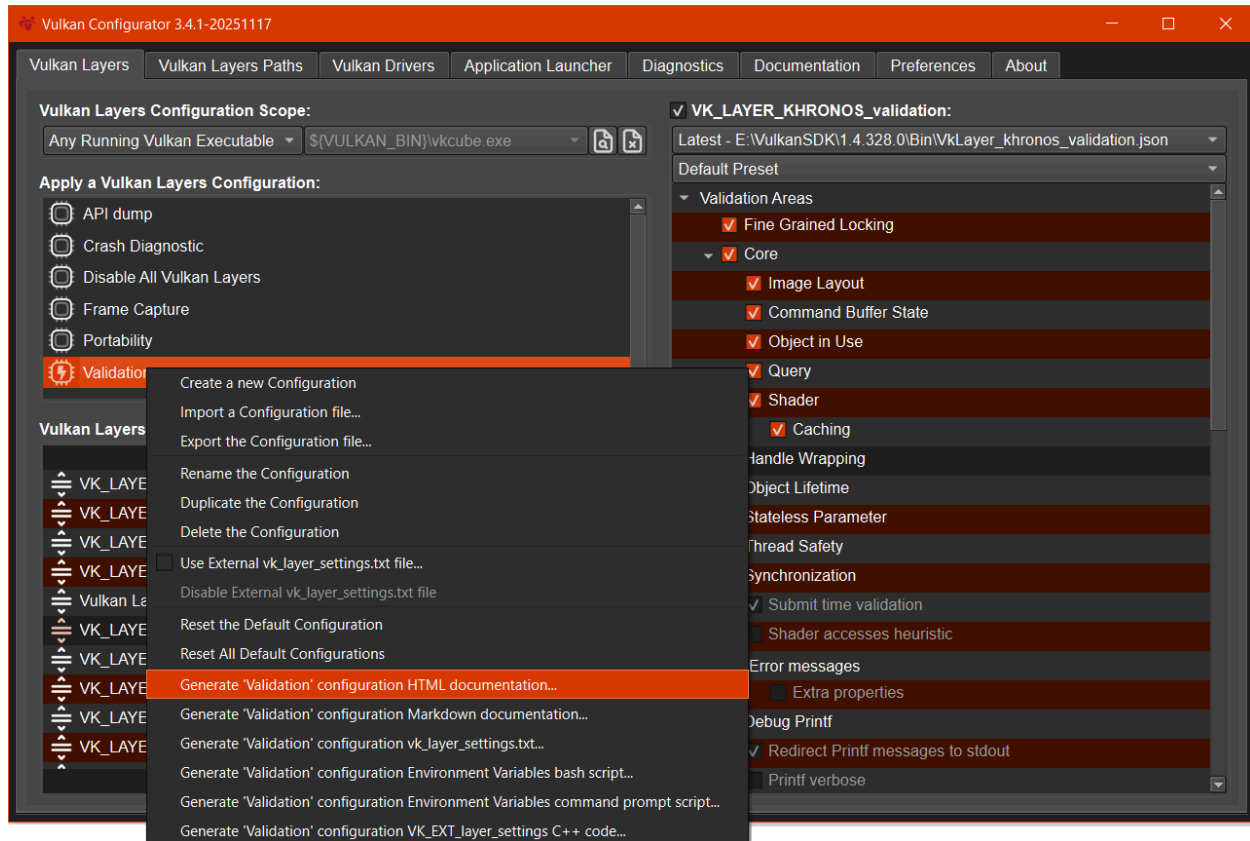
Generating Vulkan layers settings files

Generating the layer settings documentation

The layer settings reference documentation is generated from the layer manifest. It includes:

- Dependencies
- Sub-settings
- Platform support
- Links to feature documentation
- Code samples showing all three configuration methods (API, files, environment variables)

This documentation can be regenerated using the context menu of a layers configuration, it will be generated using the actual layer settings values selected and stored in this configuration.



Context menu to *generate the Validation layers configuration files*

Generating layer settings code for each method

Manually initializing layer settings with either of the three methods is possible but pretty tedious, error prone and time-consuming. This is due to the large number of settings some layers may have and considering that each new layer version may introduce new layer settings.

Vulkan Configurator solves this by letting us:

- Create and test a layers configuration with a user-interface.
- Generate ready-to-use code/scripts to initialize layer settings.

Vulkan Configurator can generate for a selected layers configuration:

- C++ helper libraries implementing `VK_EXT_layer_settings` to be used in the Vulkan application, working with either `vulkan.h` or `vulkan.hpp`
- Environment variables scripts (for shell or command prompt)

- vk_layer_settings.txt files

Using VK_EXT_layer_settings to configure layers programmatically

The generated layer settings code is a self-contained C++ header-only helper library that can be directly included in the Vulkan application code.

The layer settings are initialized by default with the value of layers configuration used to generate the library. These values can be modified during the Vulkan application execution.

Generated C++ helper library example subset

```
//Khronos Validation Layer (WINDOWS_X86, WINDOWS_ARM, LINUX, MACOS, ANDROID)
// `VK_LAYER_KHRONOS_validation` settings for version 1.4.321
/*
The main, comprehensive Khronos validation layer.

Vulkan is an Explicit API, enabling direct control over how GPUs actually work. By design,
minimal error checking is done inside a Vulkan driver. Applications have full control and
responsibility for correct operation. Any errors in how Vulkan is used can result in a
crash.

The Khronos Validation Layer can be enabled to assist development by enabling developers to
verify their applications correctly use the Vulkan API.
*/
// For more information about the layer:
https://vulkan.lunarg.com/doc/sdk/latest/windows/khronos_validation_layer.html

struct ValidationSettingData {
    static const uint32_t VERSION = VK_MAKE_API_VERSION(1, 4, 321, 0);

    // Core (WINDOWS_X86, WINDOWS_ARM, LINUX, MACOS, ANDROID)
    // Layer setting documentation:
https://vulkan.lunarg.com/doc/sdk/latest/windows/khronos\_validation\_layer.html#validate\_core
    // For more information about the feature:
https://github.com/KhronosGroup/Vulkan-ValidationLayers/blob/main/docs/core\_checks.md
    VkBool32 validate_core = VK_TRUE;

    // Image Layout (WINDOWS_X86, WINDOWS_ARM, LINUX, MACOS, ANDROID)
    // Layer setting documentation:
https://vulkan.lunarg.com/doc/sdk/latest/windows/khronos\_validation\_layer.html#check\_image\_layout
    // This setting requires ALL of the following values:
    // - VkBool32 validate_core = VK_TRUE;
    VkBool32 check_image_layout = VK_TRUE;
```

```

...
}

// `LayerSettings` allows initializing layer settings from Vulkan application code.
struct LayerSettings {
    ApidumpSettingData api_dump;
    CrashdiagnosticSettingData crash_diagnostic;
    GfxreconstructSettingData gfxreconstruct;
    ScreenshotSettingData screenshot;
    ValidationSettingData validation;
    ProfilesSettingData profiles;
    ShaderobjectSettingData shader_object;
    Synchronization2SettingData synchronization2;

    // Can be used directly with VkLayerSettingsCreateInfoEXT
    const std::vector<VkLayerSettingEXT>& info();
};

```

Generated C++ helper library usage example

```

#include "vulkan_layer_settings.hpp"

...
{
    LayerSettings layer_settings;
    layer_settings.validate_core = VK_FALSE;

    std::vector<VkLayerSettingEXT> data = layer_settings.info();

    VkLayerSettingsCreateInfoEXT layer_settings_create_info {
        VK_STRUCTURE_TYPE_LAYER_SETTINGS_CREATE_INFO_EXT, nullptr,
        static_cast<int>(data.size()), &data[0]
    };
}

```

Using environment variables to configure layers

The generated layer settings script is generated either using the Shell syntax (for Linux/macOS) or Command Prompt syntax (for Windows). It initializes all the layer settings environment variables using the values from layers configuration used to generate the script.

```

...
#! This code was generated by Vulkan Configurator 3.4.1

#! Khronos Validation Layer
#! =====

```

```

#! VK_LAYER_KHRONOS_validation - 1.4.328 (WINDOWS_X86, WINDOWS_ARM, LINUX, MACOS, ANDROID)
#! The main, comprehensive Khronos validation layer.

#! Vulkan is an Explicit API, enabling direct control over how GPUs actually
#! work. By design, minimal error checking is done inside a Vulkan driver.
#! Applications have full control and responsibility for correct operation. Any
#! errors in how Vulkan is used can result in a crash.

#! For more information about the layer:
https://vulkan.lunarg.com/doc/sdk/latest/windows/khronos_validation_layer.html

#! Core
#! -----
#! validate_core (WINDOWS_X86, WINDOWS_ARM, LINUX, MACOS, ANDROID)
#! 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.
#! For more information about the feature:
https://github.com/KhronosGroup/Vulkan-ValidationLayers/blob/main/docs/core_checks.md

#! This setting has sub-settings:
#! - export VK_KHRONOS_VALIDATION_CHECK_IMAGE_LAYOUT=true
#! - export VK_KHRONOS_VALIDATION_CHECK_COMMAND_BUFFER=true
#! - export VK_KHRONOS_VALIDATION_CHECK_OBJECT_IN_USE=true
#! - export VK_KHRONOS_VALIDATION_CHECK_QUERY=true
#! - export VK_KHRONOS_VALIDATION_CHECK_SHADERS=true
export VK_KHRONOS_VALIDATION_VALIDATE_CORE=true

#! Shader
#! -----
#! check_shaders (WINDOWS_X86, WINDOWS_ARM, LINUX, MACOS, ANDROID)
#! This will validate the contents of the SPIR-V which can be CPU intensive
#! during application start up. This does internal checks as well as calling
#! spirv-val. (Same effect using VK_VALIDATION_FEATURE_DISABLE_SHADERS_EXT)
#! This setting has sub-settings:
#! - export VK_KHRONOS_VALIDATION_CHECK_SHADERS_CACHING=true
#! - export VK_KHRONOS_VALIDATION_DEBUG_DISABLE_SPIRV_VAL=false
#! This setting requires ALL of the following values:
#! - export VK_KHRONOS_VALIDATION_VALIDATE_CORE=true
export VK_KHRONOS_VALIDATION_CHECK_SHADERS=true
...

```

Generated Environment Variables script example subset

Using command line to generate the layer settings files

We can generate the layer settings files using the Vulkan Configurator command line.

```

$ vkconfig help settings

Name
    'settings' - Command to generate layer settings files

```

Synopsis

```
vkconfig settings
    [--generate | -g] (html | markdown | txt | bash | bat | hpp | hxx)
    [--configuration | -c] [<configuration_index> | <configuration_name> | default]
    [--layer | -l] [<layer_name> | default]
    [--output-dir | -d] <output_dir>
    [--output | -o] <output_file>
    [--dry-run]
```

Description

Generate layer settings files either for system configuration or documentation of a layers configuration.

Arguments

```
`[--generate (html | markdown | txt | bash | bat | hpp)]`
    Specify the layer settings generation mode, the default value is 'txt':
    - 'html' to generate the HTML layer settings documentation, the default filename is
      'vk_layer_settings.html'
    - 'markdown' to generate the Markdown layer settings documentation, the default filename
      is 'vk_layer_settings.md'
    - 'txt' to generate the `vk_layer_settings.txt` layer settings file, the default
      filename is 'vk_layer_settings.txt'
    - 'bash' to generate the environment variables layer settings script for 'Bash', the
      default filename is 'vk_layer_settings.sh'
    - 'bat' to generate the environment variables layer settings script for
      'command prompt', the default filename is 'vk_layer_settings.bat'
    - 'hpp' to generate the C++ layer settings helper code for vulkan.h, the default
      filename is 'Vk_layer_settings.hpp'
    - 'hxx' to generate the C++ layer settings helper code for vulkan.hpp, the default
      filename is 'vk_layer_settings.hpp'

    (Run 'vkconfig layers --list' to enumerate the available layers.)

`[--configuration [<configuration_index> | <configuration_name> | default]]`
    Specify the configuration name or index in the configuration list. If the argument is
    not set or set to 'default', the default layer settings will be used.

    (Run 'vkconfig loader --list' to enumerate the available configurations.)

`[--layer <layer_name>]`
    Specify the layer name, if the argument is not set or set to 'default', all the found
    layers will be used.

    (Run 'vkconfig layers --list' to enumerate the available layers.)

`[--output-dir | -d] <output_dir>`
    Specify the output directory path. The filename used will be the default filename if
    <output_file> is not set

    - If the 'generate' is set to 'html', the default filename is 'vk_layer_settings.html'
    - If the 'generate' is set to 'markdown', the default filename is 'vk_layer_settings.md'
    - If the 'generate' is set to 'txt', the default filename is 'vk_layer_settings.txt'
```

```

- If the 'generate' is set to 'bash', the default filename is 'vk_layer_settings.sh'
- If the 'generate' is set to 'bat', the default filename is 'vk_layer_settings.bat'
- If the 'generate' is set to 'hpp', the default filename is 'vk_layer_settings.hpp'
- If the 'generate' is set to 'hxx', the default filename is 'vk_layer_settings.hpp'

`[--output | -o) <output_file>`

Specify the output file path. If <output_dir> is set, then <output_file> must be the
filename only.

- If the 'generate' is set to 'html', the default filename is 'vk_layer_settings.html'
- If the 'generate' is set to 'markdown', the default filename is 'vk_layer_settings.md'
- If the 'generate' is set to 'txt', the default filename is 'vk_layer_settings.txt'
- If the 'generate' is set to 'bash', the default filename is 'vk_layer_settings.sh'
- If the 'generate' is set to 'bat', the default filename is 'vk_layer_settings.bat'
- If the 'generate' is set to 'hpp', the default filename is 'vk_layer_settings.hpp'
- If the 'generate' is set to 'hxx', the default filename is 'vk_layer_settings.hpp'

`[--dry-run]`

Run without affecting the system and Vulkan Configurator files.

```

Vulkan Configurator command line to generate layer settings files

Vulkan SDK generated layer settings files

The Vulkan SDK has historically shipped with a `vk_layer_settings.txt` file with all the layer settings listed with their default values. This file can easily be copied by the Vulkan developer and edited manually.

An improvement is to add more documentation for each setting in the document so that the Vulkan developer doesn't have to look at the HTML documentation to figure out the dependencies, sub-settings, etc. The file is self-documented.

With Vulkan SDK 1.4.335, the Vulkan SDK includes generated code for each layer settings approaches:

- `vulkan_layer_settings.hpp` sitting next to `vulkan.h` in the SDK which is an helper C++ library with all the layer settings initialized to the layer settings default values.
- `vk_layer_settings.sh` and `vk_layer_settings.bat` sitting next to `vk_layer_settings.txt` in the SDK which are scripts, for Bash and Command Prompt respectively, which initialize all the layer settings with the default values.

These files are generated using the following commands:

vk_layer_settings.txt

```
$ VK_LAYER_PATH=$VULKAN_SDK_BUILD/build/Bin  
$ vkconfig settings --txt default -o ./vk_layer_settings.txt --dry-run
```

vk_layer_settings.sh

```
$ VK_LAYER_PATH=$VULKAN_SDK_BUILD/build/Bin  
$ vkconfig settings --bash default -o ./vk_layer_settings.sh --dry-run
```

vk_layer_settings.bat

```
$ VK_LAYER_PATH=$VULKAN_SDK_BUILD/build/Bin  
$ vkconfig settings --bat default -o ./vk_layer_settings.bat --dry-run
```

vulkan_layer_settings.hpp

```
$ VK_LAYER_PATH=$VULKAN_SDK_BUILD/build/Bin  
$ vkconfig settings --hpp default -o ./vulkan_layer_settings.hpp --dry-run
```

Revision History

Revision Date	SDK Release	Comments
May 2026	SDK 1.4.3XX.0	- Add Vulkan layer settings helper generation for vulkan.hpp - Update VK_INSTANCE_LAYERS usage
December 2026	SDK 1.4.335.0	- Update for Vulkan Configurator 3 and Vulkan Loader vk_loader_settings.json - Add Vulkan layer settings file generation
April 2024	SDK 1.3.280.0	- Fix VK_EXT_layer_settings usage example.
January 2024	SDK 1.3.275.0	- Initial release.

Vulkan and the Vulkan logo are registered trademarks of the Khronos Group Inc.