Vulkan Validation Layer
Error Reporting Improvements

Mark Lobodzinski, LunarG
August 2020

Introduction

The Vulkan validation layers provide important information about the behavior and correctness of applications, and it is important that validation transfers this information to the user as clearly, usefully, and completely as possible.

Community feedback, received through LunarXchange, Github, and Vulkan Ecosystem Surveys have highlighted several areas for improvement and we have recently turned our focus towards updating the layer messaging systems.

Recent Improvements

This document will introduce the user to several recent improvements or additions to Validation layer messaging:

- VUID-Annotated specifications in SDKs
- Returning Relevant Vulkan Objects in Debug Callback
- Unique Message Identifiers
- Plain-text Object Types and Names
- Message Filtering
- Message Repetition Limit

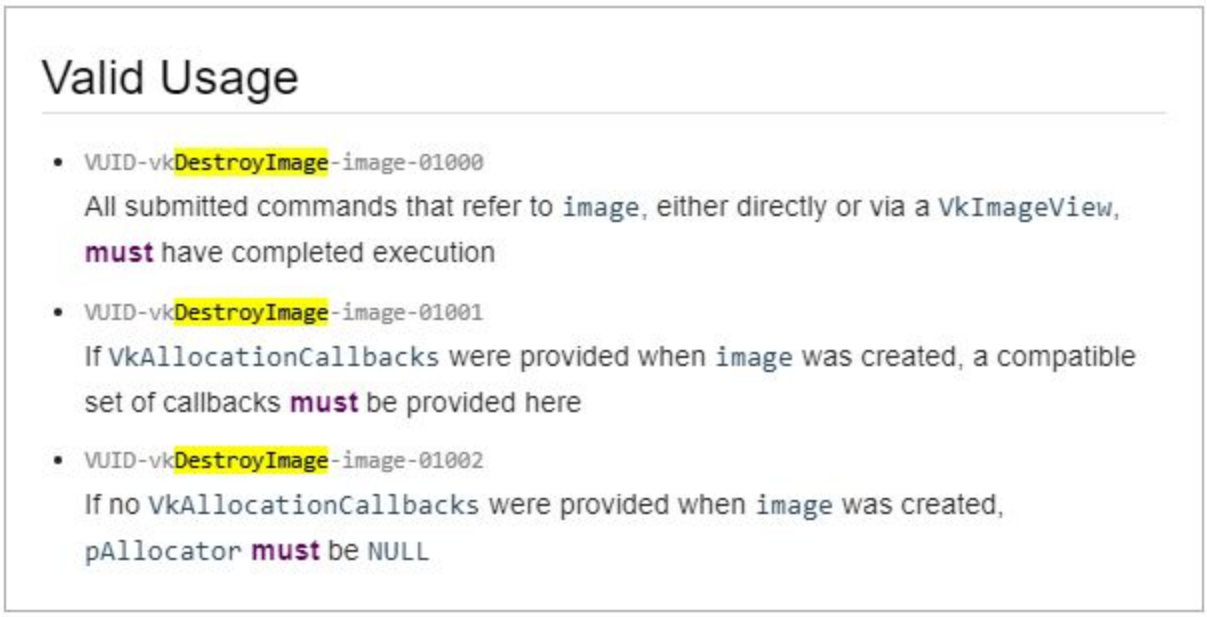
VUID-Annotated specifications in SDKs

Validation error messages have long included a link to the valid usage identifier (VUID) in the Khronos specification. This had two disadvantages:

- The link pointed to the most recent version of the specification in the Khronos Registry, instead of the version specific to the validation error
- While the links went to the area of the specification containing the VUID, it was unclear which text on that page related directly to the VUID in question

To improve the accuracy and relevance of these links, layers included in Vulkan SDK releases now provide links to annotated versions of the Vulkan specification. These documents contain plain-text VUIDs as part of the VUID text and are persistent links to the *specific version and*

class of the specification containing the VUID. Here's an example from the `DestroyImage` section of the specification:



Valid Usage

- VUID-vkDestroyImage-image-01000
  All submitted commands that refer to image, either directly or via a VkImageView, **must** have completed execution
- VUID-vkDestroyImage-image-01001
  If VkAllocationCallbacks were provided when image was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyImage-image-01002
  If no VkAllocationCallbacks were provided when image was created, pAllocator **must** be NULL

Note that the specific VUID included in the validation error message is clearly visible and gives the user immediate context and clarity as to the source of the violation.

Returning Relevant Vulkan Objects in Debug Callback

The layers were originally developed using the now-deprecated VK_EXT_debug_report extension. The debug callback definition for this extension allowed only a single Vulkan object to be returned to the user. This extension has been superseded by the VK_EXT_debug_utils extension, and its debug callback allows all relevant Vulkan objects to be returned via the callback function.
The validation layers have been updated and scrubbed to ensure that any object output in the text of a message is now also passed to the user through the callback, allowing programmatic access to these objects for debugging purposes.

```
typedef struct VkDebugUtilsMessengerCallbackDataEXT {
    VkStructureType                              sType;
    const void*                                  pNext;
    VkDebugUtilsMessengerCallbackDataFlagsEXT    flags;
    const char*                                  pMessageIdName;
    int32_t                                      messageIdNumber;
    const char*                                  pMessage;
    uint32_t                                     queueLabelCount;
    const VkDebugUtilsLabelEXT*                  pQueueLabels;
    uint32_t                                     cmdBufLabelCount;
    const VkDebugUtilsLabelEXT*                  pCmdBufLabels;
    uint32_t                                     objectCount;
```

```
    const VkDebugUtilsObjectNameInfoEXT*    pObjects;
} VkDebugUtilsMessengerCallbackDataEXT;
```

Each of the Vulkan objects returned will also include the object name/label, if specified by the user, for instance by the v`kSetDebugUtilsObjectNameEXT()` function also provided by the extension.

```
typedef struct VkDebugUtilsObjectNameInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkObjectType       objectType;
    uint64_t           objectHandle;
    const char*        pObjectName;
} VkDebugUtilsObjectNameInfoEXT;
```

Unique Message Identifiers

Validation message output now includes a hash of the VUID string, allowing simpler handling of specific messages in message callbacks or debugging utilities. This numeric value is printed out as part of the normal message output, and is also included in the callback data as the `messageIdNumber` member of the `VkDebugUtilsMessengerCallbackDataEXT` structure, or the `location` member of `PFN_vkDebugReportCallbackEXT()`.

The identifier appears in all validation messages, for instance in this example:

> Validation Error: [VUID-VkImageResolve-dstImage-00276] Object 0: handle = 0x2aa03237048, type = VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0x3ba5830000000006, type = VK_OBJECT_TYPE_IMAGE; | **MessageID = 0x3c65a6c9** | vkCmdResolveImage(): dstImage (VkImage 0x3ba5830000000006[]) is 1D but pRegions[0] dstOffset.y (0) is not 0 or extent.height (2) is not 1. The Vulkan spec states: If the calling command's dstImage...

Plain-text Object Types and Names

The validation messages previously output the Vulkan object type with the handle, but as an integer enumeration value. The messaging system now outputs the enumeration name with the object, for example:

> Validation Error: [VUID-VkImageResolve-srcImage-00268] Object 0: **handle = 0x2aa03237048, type = VK_OBJECT_TYPE_COMMAND_BUFFER**; Object 1: **handle = 0x983e60000000003, type = VK_OBJECT_TYPE_IMAGE**; Object 2: **handle = 0xa540ac0000000009, type = VK_OBJECT_TYPE_IMAGE**; | MessageID = 0x7711e6f5 | vkCmdResolveImage(): pRegions[0] baseArrayLayer must be 0 and layerCount must be 1 for all subresources if the src or dst image is 3D...

Message Filtering

The validation layers have recently been updated with a layer option that allows specific validation messages to be suppressed, should this be desired.
This can be done through the standard layer settings interfaces, by specifying the error message identifier or identifiers with the option key. For example, if the user wishes to prevent output of two messages with messageIdNumbers of 3012204 and 0x02044177, they could do so through the Vulkan Configurator utility included in the SDK, or manually by:

Vk_layer_settings.txt file

```
khronos_validation.message_id_filter=3012204,0x02044177
```

Environment Variable

Windows: `set VK_LAYER_MESSAGE_ID_FILTER=3012204;0x02044177`
Linux: `export VK_LAYER_MESSAGE_ID_FILTER=3012204:0x02044177`

Message Repetition Limit

Another often-requested feature was the ability to limit the number of times any single validation message would be reported. This option was recently implemented and allows the user to set a maximum for validation message repetitions. For example, to cap the number of message repeats to 10, the user might do so through the Vulkan Configurator utility included in the SDK, or manually by:

Vk_layer_settings.txt file

```
khronos_validation.duplicate_message_limit=10
```

Environment Variable

Windows: `set VK_LAYER_DUPLICATE_MESSAGE_LIMIT=10`
Linux: `export VK_LAYER_DUPLICATE_MESSAGE_LIMIT=10`

Acknowledgments

Please acknowledge the extensive contributions of

- Mike Schuchardt
- Tony Barbour

Revision history

Revision Date	SDK Release	Comment
September 2020	SDK 1.2.148.1	Initial release date