The Repackaged Windows Vulkan SDK

Karen Ghavam, LunarG
Richard Wright, LunarG
November 2021

Introduction

The Windows SDK released September 9, 2021 (1.2.189.2) was repackaged in a new, modular format using the Qt Installer framework. The previous SDKs were packaged using the NSIS (Nullsoft Scriptable Install System) framework.

What motivated the repackaging of the Windows SDK? What are the benefits to the user?

What motivated the repackaging of the Windows SDK?

The two primary motivators for the repackaged SDK were modularity to 2) manage SDK size and 2) get better metrics on SDK component usage.

The previous SDK packaging version did not provide a mechanism to split out components of the SDK into core (required) and optional packages. This limitation caused the SDK to grow in size for all SDK users as optional components were being added.

A prime example of this was the debug versions for the shader libraries. When using Visual Studio to link a debug version of your application with the shader toolchain static libraries included in the SDK, you will need debug versions of these static libraries to link successfully. If users were using the static libraries they would also need these debug versions of the static libraries to do their development. The size of these debug libraries was not insignificant and hence it is desirable to not burden all SDK users with their download size if they are not using the static versions of the shader toolchain libraries.

What are the Core and Optional Packages?

With the initial release of the repackaged SDK, the existing SDK components were partitioned into the following packages:

1. Core
    a. Vulkan Validation Layer
    b. Vulkan Configurator (vkconfig)
    c. Vulkan Installation Analyzer (vkvia)
    d. Vulkaninfo
    e. vkcube

   f. vk.xml
   g. Apidump
   h. Monitor
   i. Screenshot
   j. Devsim
   k. Synchronization2 Emulation Layer
   l. Shader toolchain executables
   m. Shader toolchain API libraries
   n. Gfxreconstruct
2. Optional packages
   a. 32-bit binaries for the core components
   b. Third-party tools (SDL (both 32/64-bit versions) and GLM
   c. Debuggable shader API libraries
   d. 32-bit debuggable shader API libraries

Currently, all the SDK documentation is available at vulkan.lunarg.com (with a button to zip it together and download it). In the future, an optional package will be added for the SDK documentation.

What are the benefits to the user?

There are several key benefits to using the new repackaged Windows Vulkan SDK:

- The size of the downloaded SDK can be reduced
- Additional features are available through the Qt installer
- The installer now correctly and consistently updates the register

Modular format for better management of SDK size over time

The primary benefit to the user is that they can reduce download sizes by only selecting the components of interest. This will become more important to users as LunarG continues to enhance the SDK with additional features.

The Qt Installer Framework has a richer feature set

There are features provided by the Qt Installer framework that were not available using NSIS that could be leveraged in the future. Some examples:

1. Same technology platform across operating systems for a unified experience (note: the macOS SDK is using the Qt installer framework as well).
2. Automatic notifications of newer package versions.
3. You can install an SDK, and, later via the maintenance tool, add or remove SDK components from your installation

Correct and consistent updating of the registry

An additional benefit of the new SDK installer is that it now consistently and correctly updates the PATH and registry entries. Installing the Vulkan SDK sets the system environment variable VULKAN_SDK to the directory in which the SDK is installed. Also, the expansion of %VULKAN_SDK%\Bin is prepended to the System PATH environment variable.

There was a known defect with the old (NSIS) Windows SDK:

> When you install multiple SDKs, the PATH environment variable is set to point to the SDK that was most recently installed. The HKLM registry, however, is set up to point to the newest SDK, regardless of the order of install. This means that executables from the SDK will come from the most recently installed SDK, while layers (in the HKLM registry) will come from the newest SDK.

The result is that installing multiple SDKs did not provide a correct mechanism to have multiple SDK versions installed and to switch between them by switching your PATH environment variable.

Note: A user can install as many versions of SDKs as desired to have the binaries on their system. The recommended method to switch between SDK versions is to use the Vulkan Configurator (vkconfig). The Vulkan Configurator has a mechanism to override what is in the registry and hence you can select any version of layers from previous SDK installs.

Why is the new packaged SDK incompatible with the old SDK?

The NSIS Windows SDK installer behavior only removed layers from the registry that it “knew” and attempted to retain the newest layers in the registry. The side effect is that if a new correctly behaving SDK is installed and then an old NSIS framework SDK is installed, both sets of layers will be populated in the registry. The end result of this is that the loader will often find and use an older layer rather than the one from the most recently installed SDK.

Hence the reason for the popup during installation of the newer SDK:

> *If you have installed one or more newer SDKs (1.2.189.1 or newer), and you wish to install an older SDK prior to 1.2.189.1, you must uninstall the most recently installed SDK first. To uninstall the most recently installed SDK, use the Windows "Add and remove programs feature,” or the maintenancetool.exe in the SDK install directory.*

If a newer Windows SDK that uses the Qt Installer Framework is installed on your system, it correctly clears out explicit layers from the registry during installation and installs the new layers from the new install into the registry.