NVIDIA Streamline 2.2.1

Developer Guide

Version 1.0
October 2023

Streamline 2.2.1 New Items - Overview

DLSS Ray Reconstruction

- New Particle Layer API

DLSS Frame Generation

- New Distortion Map API
- Support for Dynamic Frame Generation which can intelligently enable/disable DLSS Frame Generation to help ensure positive performance scaling

Miscellaneous

- Bug fixes and stability enhancements

Key Steps to a Successful DLSS 3.5 Integration

1. Integrate Streamline 2.x SDK (without any features) and focus on manual hooking, resource state tracking, etc. - **1 day**

2. Check for the official NVIDIA & Streamline dual signatures on sl.interposer.dll before loading the DLL - **30 mins**

3. Check for system (hardwares/software) support for each of the DLSS 3 features - and show appropriate error messages to end users based on reported hw support - **1 hour**

4. Integrate DLSS Super Resolution via Streamline - pass in the necessary input resources and set up the upscaling pipeline (before all post-processing) - **2 days**

5. Validate IQ and performance benefits from DLSS Super Resolution - **1 day**

6. Integrate DLSS Ray Reconstruction via Streamline - **4 days**

7. Validate IQ and performance benefits from DLSS Ray Reconstruction - **1 day**

8. Integrate Reflex via Streamline - set up the appropriate markers, sleep call, etc. - **1 day**

9. Validate that input latency gets reduced, using FrameView SDK or GFE in game overlay or the Reflex validation HUD - **4 hours**

10. Integrate DLSS Frame Generation - pass in the appropriate constants, camera matrices, and input resources in addition to the ones marked for DLSS Super Resolution (e.g. hudless and ui color/alpha). Be sure to also disable DLSS Frame Generation when appropriate, such as when in-menu or scene transitions - **2 days**

11. Validate that inputs are correct using Streamline Imgui plugin and buffer visualization using the development DLLs - **1 day**

12. Once IQ and perf benefits from DLSS Frame Generation are validated, replace the watermarked DLLs with non-watermarked, production ready DLLs from NV - **1 day**

    Total Estimated Time: **~14 days**

DLSS Super Resolution

*Detailed information can be found in the **Streamline DLSS Programming Guide** or the exhaustive **Complete DLSS Programming Guide***

Basic Integration Checklist

- ☐ Game-specific Application ID is used during initialization
    - In the absence of this, future hardware support, bug fixes and OTA updates for DLSS Super Resolution might not work
- ☐ Make sure that DLSS Super Resolution is integrated as close to the start of post-processing as possible
    - If DLSS SR is integrated after post-processing, it can lead to image quality artifacts or other visual issues with post-processing effects like Bloom, etc.
- ☐ Mip-map bias set when DLSS is enabled
    - Without this, textures will look blurry / smudgy / low-resolution
- ☐ Motion vectors for all scenes, materials and objects are accurate
    - Incorrect mvecs can lead to increased ghosting and other IQ issues
- ☐ Static scenes resolve and compatible jitter confirmed (see section 8.3)
    - Incorrect / missing jitter can lead to poor anti-aliasing and increased flickering
- ☐ Exposure value is properly sent each frame (or auto-exposure is enabled)
    - Improper exposure can cause increased ghosting, flickering, and other IQ issues
- ☐ DLSS modes are queried and user selectable in the UI and/or dynamic resolution support is active and tested.
    - The modes allow end users better control over the perf vs IQ tradeoff. If all modes are not queried and shown, end users will not get that choice
- ☐ Full production non-watermarked DLSS library (nvngx_dlss.dll) is packaged in the release build
    - Don't ship a nvidia confidential watermark in your game
- ☐ Pass in Camera Reset Flag on scene changes, view changes (first person to third person), or during camera jumps in cutscenes.
- ☐ Perform NGX cleanup / shutdown procedures when DLSS is no longer needed.
    - Otherwise, you'll leak resources / memory

IMPORTANT: *DLSS should only replace the primary upscale pass on the main render target and should not be used on secondary buffers like shadows, reflections etc.*

DLSS Ray Reconstruction

*Detailed information can be found in the **Streamline DLSS-RR Programming Guide** or the exhaustive **DLSS-RR Integration Guide** both of which are available as part of the DLSS-RR SDK Package.The DLSS-Ray Reconstruction quick start guide is very similar to the DLSS Super Resolution quick start guide.*

Basic Integration Checklist

- ☐ Game-specific Application ID is used during initialization
    - In the absence of this, future hardware support, bug fixes and OTA updates for DLSS features might not work
- ☐ DLSS-RR is an addition to DLSS Super Resolution. It uses the same Perf Quality modes that have been set for DLSS Super Resolution. So make sure DLSS Super Resolution is integrated first.
- ☐ Make sure that both DLSS Ray Reconstruction and DLSS Super Resolution are integrated at the start (or as close to the start) of post-processing as possible.
    - If either are integrated after post-processing, it can lead to image quality artifacts or other visual issues with post-processing effects like Bloom, etc.
- ☐ Make sure other denoisers in your pipeline such as NRD are completely disabled.
- ☐ Mip-map bias set when DLSS RR is enabled.
    - Without this, textures will look blurry / smudgy / low-resolution
- ☐ Other required buffers are provided as mentioned in section 3.4 of the DLSS RR Programming Guide
    - Diffuse and Specular Albedo, Normals,, Roughness, Input Color, Motion Vectors, Depth, Specular Motion Vectors (or Specular Hit distance with corresponding matrices)
- ☐ DLSS RR Requires Linear Depth to be provided which is different from those provided to DLSS Super Resolution and DLSS Frame Generation
    - Please use kBufferTypeLinearDepth specifically provided for this
    - Please set the Inverted Depth bit if the Depth buffer provided has inverted z ordering
- ☐ Motion vectors / Specular Motion Vectors for all scenes, materials and objects are accurate
    - Incorrect Motion Vectors can lead to increased ghosting and other IQ issues
- ☐ Static scenes resolve and compatible jitter confirmed (see section 8.3 of the DLSS Programming Guide)
    - Incorrect / missing jitter can lead to poor anti-aliasing and increased flickering
- ☐ DLSS modes are queried and user selectable in the UI and/or dynamic resolution support is active and tested.
    - The modes allow end users better control over the perf vs IQ tradeoff. If all modes are not queried and shown, end users will not get that choice
- ☐ A DLSS RR enable / disable toggle is available
    - Generally this should be made available when Ray Tracing or Path Tracing is being used
    - Ray Tracing and Path Tracing are available in DX12 and Vulkan only.

- [ ] Full production non-watermarked DLSS Ray Reconstruction library (nvngx_dlssd.dll) is packaged in the release build
    - Don’t ship a nvidia confidential watermark in your game
- [ ] Pass in Camera Reset Flag on scene changes, view changes (first person to third person), or during camera jumps in cutscenes.
- [ ] Perform NGX cleanup / shutdown procedures when DLSS is no longer needed.
    - Otherwise, you’ll leak resources / memory

DLSS Frame Generation

*Detailed information can be found in the **DLSS FG Programming Guide***

Basic Integration Checklist

- ☐ All the required inputs are passed to Streamline: *depth buffers, motion vectors, HUD-less color buffers, UI color buffer*
  - If any of the input buffers are missing or incorrect, you'll get severe IQ artifacts such as blurry objects, broken / disjointed characters on screen, increased flickering and instability in UI and HUD elements, etc.
  - Use the debug visualization to check that (1) buffers are passed in and (2) they are frame-aligned
- ☐ Common constants and frame index are provided for each frame using slSetConstants and slSetFeatureConstants methods (Streamline 2.0.1-specific APIs)
  - Incorrect values here will lead to a mismatch between FG and Reflex integration, and will cause magenta tinted image or lead to increased input latency and stutter
- ☐ All tagged buffers are valid at frame present time, and they are not reused for other purposes
  - Reusing of tagged buffers before present time can lead to severe corruption in the entire image (or at least parts of it, depending on the amount of the resource that's been reused)
- ☐ Buffers to be tagged with unique viewport id 0
  - DLSS Frame Generation requires depth, motion vectors and HUD-less color buffers which are used during the SwapChain::Present call so if these buffers are going to be reused, destroyed or changed in any way before the frame is presented their life-cycle needs to be specified correctly.
  - DLSS FG does not support multiple viewports (unlike DLSS Super Resolution and other features supported by SL SDK), so all input buffers for DLSS Frame Generation must use a viewport id of 0
- ☐ Make sure that frame index provided with the common constants is matching the presented frame
  - Incorrect values here will lead to a mismatch between FG and Reflex integration, and will cause magenta tinted image or lead to increased input latency and stutter
- ☐ Inputs are passed into Streamline look correct as well as camera matrices and dynamic objects
  - Incorrect inputs will lead to severe IQ issues such as blurry / smudgy objects, increased ghosting, disjointed characters, flickering UI, and generally garbled images
  - Dynamic motion should mark any object moving independently of the player camera motion. This heuristic helps identify a correct camera matrix format.
  - The camera matrix should not have jitter information.
  - If the MVECs are jittered, set the MVECJittered SL boolean or NGX Equivalent.
- ☐ Application checks the signature of sl.interposer.dll to make sure it is a genuine NVIDIA library

    - In the absence of this, a malicious user can inject their own version of sl.interposer.dll, allowing them to hijack the rendering pipeline (can be used by cheaters, hackers, or worse)
- ☐ Requirements for Dynamic Resolution are met (if the game supports Dynamic Resolution)
    - DLSS Frame Generation supports dynamic resolution of the MVec and Depth buffer extents. Dynamic resolution may be done via DLSS or an app-specific method
    - Since DLSS Frame Generation uses the final color buffer with all post-processing complete, the color buffer must be a fixed size -- it cannot resize per-frame
    - When DLSS Frame Generation dynamic resolution mode is enabled, the application can pass in a differently-sized extent for the MVec and Depth buffers on a per frame basis. This allows the application to dynamically change its rendering load smoothly.
- ☐ Disable DLSS Frame Generation (by setting sl::DLSSGOptions::mode to eDLSSGModeOff) when the game is paused, loading, in menu and in general NOT rendering game frames and also when modifying resolution & full-screen vs windowed mode
    - This will ensure the best overall gaming stability and experience
- ☐ Reduce the amount of motion blur; when DLSS Frame Generation enabled, halve the distance / magnitude of motion blur
    - Incorrect motion blur can lead to IQ artifacts where the motion blur is essentially doubled due to additional generated frames
- ☐ Reflex is properly integrated (see checklist in Reflex Programming Guide)
    - Proper integration of FG is absolutely dependent on a proper integration of Reflex also. If the Reflex integration is incorrect, expect to see significant increases in input latency when FG is enabled, along with a magenta tinted image
- ☐ In-game UI for enabling/disabling DLSS Frame Generation is implemented and follows RTX UI Guidelines
    - Make operation intuitive and uniform for end users
- ☐ Only full production non-watermarked libraries are packaged in the release build
    - Otherwise you'll see a message on the top left of your screen that says "NVIDIA CONFIDENTIAL - PROVIDED UNDER NDA - DO NOT DISTRIBUTE IN ANY WAY"
- ☐ No errors or unexpected warnings in Streamline and DLSS Frame Generation log files while running the feature
    - Different errors and warnings can have different consequences including possibility of crashes, IQ artifacts and perf slowdowns.
- ☐ Pass in Camera Reset Flag on scene changes, view changes (first person to third person), or during camera jumps in cutscenes.
- ☐ Semi-transparent objects require some care and attention; Do their motion require tracking, or the motion of the objects seen through them. If it is their motion (ie. a gun scope not raised to the eye), then mvecs and depth values should be provided for them. This may require extra coding not normally required for transparent objects.
- ☐ When calculating in-game FPS, always account for the extra frames generated by DLSS-FG. This can be queried, per frame, and is returned by

DLSSGState::numFramesActuallyPresented, this will either be 1 or 2 depending on whether DLSS Frame Generation was enabled for that given frame.

- If calculating real-time FPS by taking the time between present calls, then the time will need dividing by this value.
- If calculating average FPS by dividing the total time by the number of game frames presented, then the DLSSGState::numFramesActuallyPresented will need to accumulate over the time period and will represent the total number of frames actually presented.

Reflex

*Detailed information can be found in the **SL Reflex Programming Guide***

Basic Integration Checklist

- ☐ Reflex Low Latency default state is “On”
- ☐ All Reflex modes (Off, On, On + Boost) function correctly
- ☐ PC Latency (PCL) in the Reflex Test Utility is higher than zero
  - Zero means that some of the markers were not integrated correctly
  - You can visualize marker placement in NSight Systems to help aid in debugging, including seeing the sim marker moving to JIT when Reflex is enabled vs. disabled.
- ☐ Reflex Test Utility Report has passed without warning with DLSS Frame Generation enabled
  - Indicates that Reflex worked correctly in On / Low Latency mode. Your input latency should have gone down by at least 20%
- ☐ Reflex Test Utility Report has passed without warning with Reflex set to “On”
- ☐ Reflex does not significantly impact average framerate (fps) more than 4% when Reflex is enabled
  - Reflex set to “On + Boost” is expected to have a slightly larger performance impact in order to provide the absolute lowest latency
- ☐ Reflex Test Utility Report has passed without warning with Reflex set to “On + Boost”
- ☐ Reflex Markers are always sent regardless of Reflex Low Latency mode state
- ☐ Reflex Flash Indicator appears when left mouse button is pressed
- ☐ Reflex UI settings are following the UI Guidelines
- ☐ Keybinding menus work properly (no F13)
- ☐ PC Latency (PCL) is higher than zero on non-NVIDIA GPUs
- ☐ Reflex UI settings are disabled or not available on non-NVIDIA GPUs

Reflex Integration Checklist Steps

1. Download **Reflex Verification Tools**
2. Install FrameView SDK
    - Double click the FrameView SDK Installer (**FVSDKSetup.exe**)
    - Restart system
3. Run **ReflexTestSetup.bat** from an administrator mode command prompt
    - This will force the Reflex Flash Indicator to enable, enable the Verification HUD, set up the Reflex Test framework, and start ReflexTest.exe.
4. Check Reflex Low Latency modes
    - Run game
        - Make sure game is running in fullscreen exclusive
        - Make sure VSYNC is **disabled**
        - Make sure MSHybrid mode is not enabled
    - Ensure Reflex Low Latency mode is set to “On” in game GUI and in the Verification HUD
        - Use the Reset/Default button in UI if it exists
    - Cycle through the Reflex modes in UI and confirm it matches with “Reflex Mode” within the Verification HUD
5. Running Reflex Tests
    - For titles supporting DLSS Frame Generation, **enable** DLSS-FG
        - Press **Alt + T** in game to start the test (2 beeps)
        - Analyze results after the test is done (3 beeps)
            1. Test should take approximately 5 minutes
            2. Look for "PASSED" and check for warnings in ReflexTest.exe output
    - **Disable** DLSS-FG (if available) and set Reflex to **“On”**
        - Press **Alt + T** in game to start the test (2 beeps)
        - Analyze results after the test is done (3 beeps)
            1. Test should take approximately 5 minutes
            2. Look for "PASSED" and check for warnings in ReflexTest.exe output
    - Set Reflex to “**On + Boost**”
        - Press **Alt + T** in game to start the test (2 beeps)
        - Analyze results after the test is done (3 beeps)
            1. Test should take approximately 5 minutes
            2. Look for "PASSED" and check for warnings in ReflexTest.exe output
6. Test that markers are always sent even when Reflex is Off
    - Set Reflex to “**Off**”
    - Look at the Verification HUD and ensure the markers are still updating
7. Test the Reflex Flash Indicator
    - Verify the Reflex Flash Indicator is showing
        - Notice the gray square that flashes when the left mouse button is pressed

        - Verify that the Flash Indicator in the Verification HUD increments by 1 when the left mouse button is pressed
8. Check UI
    - Verify UI follows the Guidelines
9. Check keybinding
    - Run **capturePclEtw.bat** in administrator mode from command prompt
    - Go back to the game. Start gameplay
    - Check the Keybinding menu to make sure F13 is not being automatically applied when selecting a key
    - Go back to the command prompt and press any key to exit the bat
10. Run **ReflexTestCleanUp.bat** in administrator mode from command prompt
    - This disables the Reflex Flash Indicator and the Reflex Test framework
11. Test on other IHV (if available)
    - Install other IHV hardware
    - Install FrameView SDK and restart the system
    - Run **PrintPCL.exe** in administrator mode command prompt
    - Run game
    - Press **Alt + t** in game
        - Look at the PCL value in the command prompt. If the value is not 0.0, then PCL is working
    - Press **Ctrl + c** in the command prompt to exit **PrintPCL.exe**
    - Check to make sure Reflex UI is not available
12. Send NVIDIA Reflex Test report and Checklist results
    - Email results to NVIDIA alias: reflex-sdk-support@nvidia.com

FAQ

Streamline

Q: What do I need to do to ensure third party overlays (STEAM, Epic Game Store, etc) work correctly with Streamline?

A: The following rules should be followed in order for 3rd party overlays to work correctly with Streamline:

- Overlays in general **must not make assumptions about swap-chain and command queues, when DLSS Frame Generation is active there could be multiple command queues and multiple asynchronous presents**.
- Overlays should intercept `IDXGIFactory::CreateSwapChainXXX` to obtain the correct swap-chain and command queue used to present frames.
- If integrated in engine, overlays should initialize **before** `slInit` is called
- There is an optional flag `sl::PreferenceFlags::eUseDXGIFactoryProxy` which can be used to avoid injecting SL hooks in DXGI factory v-table, this might help resolve some issues with overlays.

Q: What do I do if the D3D debug layer is complaining about incorrect resource states?

A: Please provide correct state when tagging Streamline resources

Q: What do I do if I get a crash in Swapchain::Present or some similar unexpected behavior?

A: Please double check that you are **NOT** linking dxgi.lib/d3d12.lib together with the sl.interposer.dll. If this is not an issue, please view the Streamline logs for further guidance on what the cause could be.

Q:Should I enable Streamline on non-NVIDIA hardware?

A: Yes. While Streamline is an SDK developed by NV, it supports multiple features that work across all hardware vendors (for example: NVIDIA Image Sharpening (NIS), NVIDIA Real-Time Denoiser (NRD), Reflex (PCL))

Q:Streamline 2.1 introduces OTA functionality for Streamline. What sort of things can be updated OTA and how is this different from earlier DLSS OTA?

A: Streamline 2.1 expands over-the-air (OTA) functionality for all Streamline plugins. Previously this OTA was limited to DLSS updates only. . Significant improvements have been made, including performance, smoothness, and memory optimizations as well as bug fixes and security enhancements. Streamline OTA is supported on NVIDIA GPUs.

DLSS Super Resolution

Q: DLSS Super Resolution output does not look correct. What went wrong?

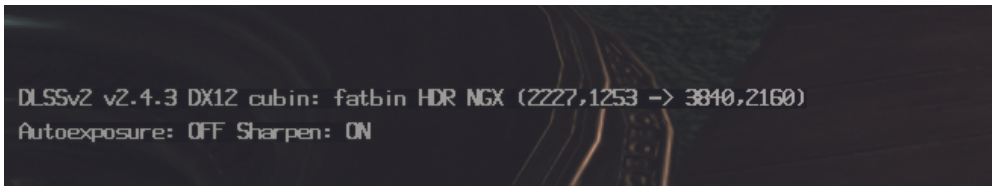
A: If the DLSS output does not look right (*ie: instability when moving the camera*) please check the following items. Also, review see section 8 of the DLSS Super Resolution Programming Guide for detailed troubleshooting information:

- If your motion vectors are in pixel space then scaling factors sl::Constants::mvecScale should be {1 / render width, 1 / render height}
- If your motion vectors are in normalized -1,1 space then scaling factors sl::Constants::mvecScale should be {1, 1}
- Make sure that jitter offset values are in pixel space
- NVSDK_NGX_Parameter_FreeMemOnRelease is replaced with slFreeResources
- NVSDK_NGX_DLSS_Feature_Flags_MVLowRes is handled automatically based on tagged motion vector buffer's size and extent.

Q: How can I verify that DLSS Super Resolution is enabled?

A: You can enable the DLSS on-screen indicator utility by running “ngx_driver_onscreenindicator.reg” before starting your application. Once DLSS Super Resolution is enabled, you’ll see confirmation on the overlay in-game.

Once finished, you can run “ngx_driver_onscreenindicator_off.reg” to disable the indicator.



Q: What new features does DLSS Super Resolution 3.1.x bring compared to 2.1.x?

A: DLSS 3.1.x adds the following features:

- Preset Updates via OTA (see section 2.4)
- Model Selection API (see section 3.12)
- Querying for Feature Support Requirements (see section 2.3)
- Querying for Vulkan Extension Requirements (see section 2.3.1)
- Deprecation of DLSS sharpening (see section 3.11)

Q: What happened to DLSS Sharpening?

A: The sharpening and softening pass provided in previous versions of the SDK are now deprecated. All developers are encouraged to use the Image Scaling SDK via the DLSS SDK or directly on GitHub.

DLSS Ray Reconstruction

Q: What is DLSS Ray Reconstruction?

A: DLSS Ray Reconstruction is a new neural network for all GeForce RTX GPUs that improves the image quality of path-traced and intensive ray-traced content. Ray Reconstruction replaces hand-tuned denoisers with an NVIDIA supercomputer-trained AI network that generates higher-quality pixels in between sampled rays.

Q: I want to integrate DLSS Ray Reconstruction into my title. Is everything I need included within the Streamline 2.2 SDK?

A: DLSS Ray Reconstruction functionality is limited to an NDA Streamline 2.2 package, **and is not supported in the public SDK on Github**. Please contact your NVIDIA account manager for early access, or sign up to be notified when its publicly available here: https://developer.nvidia.com/rtx/dlss/notify-me

Q: What GeForce GPUs support DLSS Ray Reconstruction?

A: Any GeForce RTX GPU can support DLSS Ray Reconstruction as they feature the required Tensor Cores. This hardware requirement is the same as with DLSS Super Resolution.

Q: Can I enable DLSS Ray Reconstruction without enabling any other DLSS technologies?

A: DLSS Ray Reconstruction should not be enabled unless DLSS Super Resolution is enabled.

Q: Will I see a performance benefit when I enable DLSS Ray Reconstruction?

A: Game content with multiple ray traced effects, or full ray tracing, typically has several denoisers tuned for each specific type of lighting. DLSS Ray Reconstruction replaces these multiple denoisers with a single neural network, so it can run faster in heavily ray traced scenes. However, the performance benefits will vary according to the scene, settings, and GPU being used.

Q: Should Ray Reconstruction have a separate setting in game menus?

A: Please consult the RTX UI Guidelines for details.

Q: What should I do with any other denoiser I might have already integrated?

A: Please ensure that when DLSS-RR is enabled, any other Denoiser (such as NRD) is completely disabled.

Q: Does Ray Reconstruction need to be integrated at multiple places in the pipeline with specialized passes?

A: Similar to DLSS Super Sampling, Ray Reconstruction is designed to be a single point of integration

Q: What kind of Ray Tracing / Path Tracing models is Ray Reconstruction compatible with?

A: DLSS Ray Reconstruction is designed to be compatible with all Ray Tracing and Path Tracing models.

Q: Which Operating Systems is DLSS Ray Reconstruction available on ?

A: DLSS Ray Reconstruction is supported on Windows, Linux and Proton (Proton Experimental is currently required to enable DLSS Ray Reconstruction).

Q: Does DLSS Ray Reconstruction work with all DLSS Super Resolution modes ?

A: “Ultra-performance” and “DLAA” modes are not currently supported when DLSS Ray Reconstruction is enabled. All other modes for DLSS Super Resolution are supported.

DLSS Frame Generation

Q: What are the HW/SW requirements for enabling DLSS Frame Generation?

A: The following criteria must be met to enable DLSS Frame Generation:

- GeForce RTX 40 Series GPU (Desktop or Notebook)
- Minimum Windows OS version of Win10 20H1 (version 2004, build 19041 or higher)
- **Windows Hardware-accelerated GPU Scheduling (HWS) must be enabled** via Settings : System : Display : Graphics : Change default graphics settings.

Q: How do I confirm the requirements are met on a system?

A: In order for DLSS-FG to work correctly certain requirements regarding the OS, driver and other settings on a user's machine must be met. To obtain DLSS-FG configuration and check if all requirements are met you can use the following code snippet:

```
Unset
sl::FeatureRequirements requirements{};
if (SL_FAILED(result, slGetFeatureRequirements(sl::kFeatureDLSS_G,
requirements)))
{
    // Feature is not requested on slInit or failed to load, check logs,
handle error
}
else
{
    // Feature is loaded, we can check the requirements
    requirements.flags & FeatureRequirementFlags::eD3D11Supported
    requirements.flags & FeatureRequirementFlags::eD3D12Supported
    requirements.flags & FeatureRequirementFlags::eVulkanSupported
    requirements.maxNumViewports
    // and so on ...
}
```

Q: What happens if I don't tag resources correctly?

A: If validity of tagged resources cannot be guaranteed (for example game is loading, paused, in menu, playing a video cut scene etc.) all tags should be set to null pointers to avoid stability or IQ issues.

For more information, please reference Section 5.0 of the DLSS FG Programming Guide

Q: Why am I getting random crashes when my game is loading, or if I'm in a menu screen or playing a cut scene, etc.?

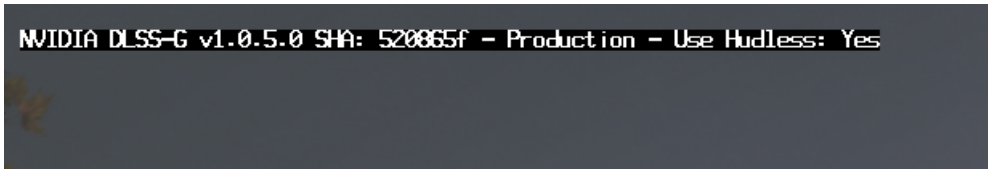
A: If validity of tagged resources cannot be guaranteed (for example game is loading, paused, in menu, playing a video cut scene etc.) all tags should be set to null pointers to avoid stability or IQ issues.

For more information, please reference Section 5.0 of the DLSS FG Programming Guide

Q: How can I verify that DLSS Frame Generation is enabled?

A: You can enable the DLSS on-screen indicator utility by running "ngx_driver_onscreenindicator.reg" before starting your application. Once DLSS Frame Generation is enabled, you'll see confirmation on the overlay in-game.

Once finished, you can run "ngx_driver_onscreenindicator_off.reg" to disable the indicator.

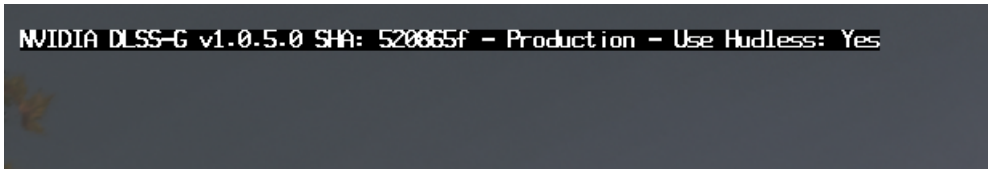


Developers can also leverage the sl.imgui.dll overlay which was introduced with Streamline 2.0.

Q: How can I verify that hudless buffers are tagged?

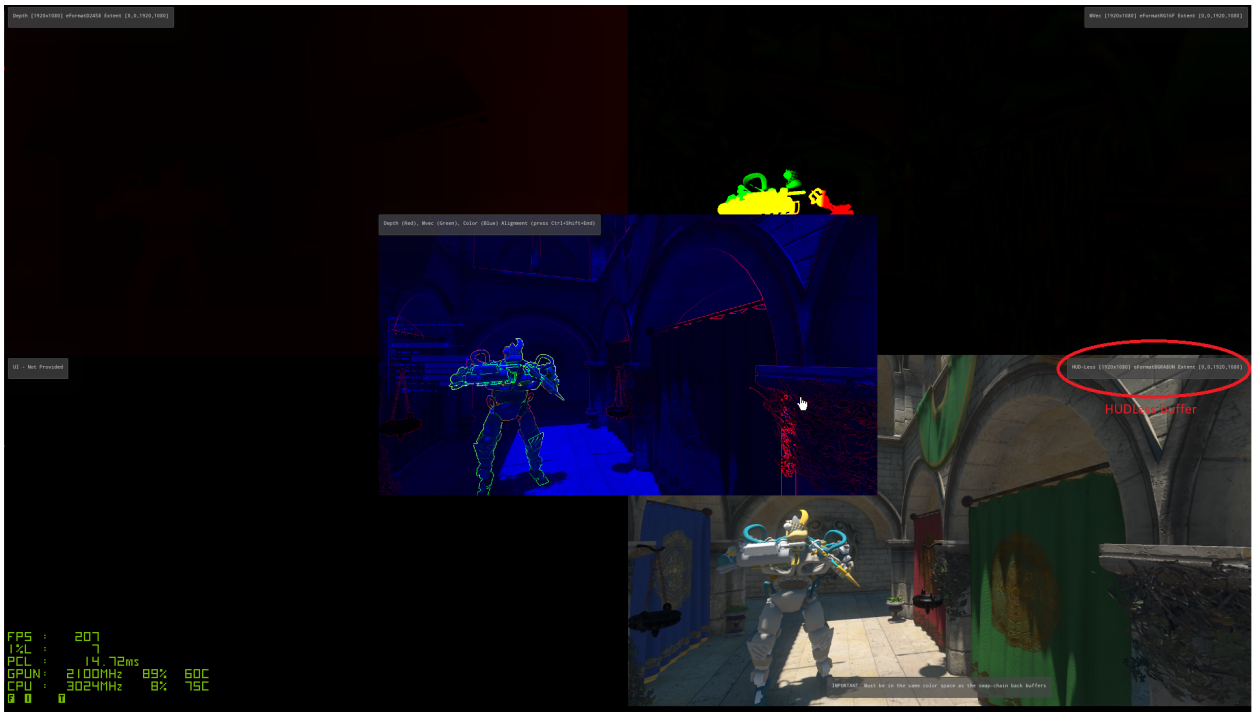
A: You can enable the DLSS on-screen indicator utility by running "ngx_driver_onscreenindicator.reg" before starting your application. Once DLSS Super Resolution is enabled, you'll see "Use Hudless: Yes" in the overlay if hudless buffers are tagged.

Once finished, you can run "ngx_driver_onscreenindicator_off.reg" to disable the indicator.



Developers can also leverage the sl.imgui.dll overlay which was introduced with Streamline 2.0.1.

1. Confirm DLSS FG is running
2. Confirm DLSS Frame Generation is running
3. Press Debug key (CTRL+SHIFT+INS)
4. Visualizes all buffers (allows dev to confirm hudless buffer is correct)
5. Confirm functionality on screen



Q: Is DLSS Frame Generation compatible with HDR?

A: Yes, DLSS Frame Generation is compatible with HDR. However, DLSS Frame Generation currently does **NOT** support FP16 pixel format and scRGB color space.

Q: When I enable DLSS Frame Generation, my performance drops. Why is that?

A: DLSS Frame Generation has a finite cost to run. It provides an fps boost if the time taken to generate a frame is less than the time taken to render the same frame through the game engine.
However, if the game renders really fast (typically >200 fps, but lower in some cases), then DLSS FG might actually slow down the game's overall framerate. Dynamic Frame Generation is designed to help with this scenario.
However, if the amount of available video memory is less than the amount needed for DLSS Frame Generation, then DLSS FG might actually slow down the game's overall framerate.

Q: I've integrated DLSS Frame Generation, but don't see any change in my performance when it's enabled. What could be the reason?

A: DLSS Frame Generation adds additional calls to Present(), which the game engine may not track by default.

In-game fps numbers, as a result may not match with numbers reported from outside the app process via tools like FrameView, Geforce Experience, CapFrameX, Windows Game Bar, etc.
The Streamline API provides information about the number of frames presented through DLSSGState::numFramesActuallyPresented, which should be hooked up to the engine's fps reporting function. This will align with fps reported by external tools as well as the game engine itself.

Q: How can I verify that my camera data and motion vectors are aligned?

A: Developers can also leverage the sl.imgui.dll overlay which was introduced with Streamline 2.0.1.

1. Confirm DLSS FG is running
2. Confirm DLSS Frame Generation is running
3. Press Debug key (CTRL+SHIFT+INS)
4. Enter “Dynamic Objects” view (cycle through views with CTRL+SHIFT+END)
5. Confirm functionality on screen

Q: If I enable VSYNC when DLSS Frame Generation is enabled, the game stutters. Why is that?

A: If you're using Streamline 2.0.1, you now have DLSS Frame Generation support for V-SYNC (*set within the NVIDIA Control Panel*) without any requirement of having a G-SYNC compatible display. In-game V-SYNC setting support is not available and should remain disabled when DLSS Frame Generation is enabled.

V-SYNC (*NV Control Panel*)	**G-SYNC** (*NV Control Panel*)	**Result**	**Required Streamline Version**
Off	Off	Max FPS, Low Latency, Tearing	1.3.3+
Off	On	Max FPS, Low Latency, Tearing above Refresh Rate	1.3.3+
On	Off	No Tearing, Not Smooth, High Latency	< 2.0.1
On	Off	No Tearing, Smooth, Lower Latency	2.0.1+
On	On	No Tearing, Smooth, Lowest Latency VSYNC ON option	1.3.3+

Starting with driver version 526.98 and higher, DLSS Frame Generation is compatible with VSYNC on G-SYNC and G-SYNC Compatible monitors and TVs. To enable this feature:

- Enable G-SYNC: NVIDIA Control Panel --> Display --> Setup G-SYNC
- Turn VSYNC On: NVIDIA Control Panel --> 3D Settings --> Manage 3D Settings

Note: DLSS Frame Generation is currently **not** compatible with VSYNC on displays that are not G-SYNC or G-SYNC Compatible when using Streamline 2.0.0a or earlier.

Q: What does the Auto Scene Change Detection feature do?

A: Auto Scene Change Detection aims to automatically prevent Frame Generation from producing difficult-to-create frames between a substantial scene change. It does this by analyzing the in-game camera orientation on every input DLSS Frame Generation frame pair. Auto Scene Change Detection simplifies integration of new DLSS 3 titles, is backwards compatible with all DLSS 3 integrations, and supports all rendering platforms.

Q: What does the Dynamic Frame Generation feature do?

A: Dynamic Frame Generation can intelligently enable/disable DLSS Frame Generation to help ensure positive performance scaling.

Q: I've enabled Dynamic Frame Generation, but I still see performance drops. Why is that?

A: Dynamic Frame Generation uses a heuristic that samples CPU and GPU performance over time. If the game's engine performance varies greatly within this sampling period, then Dynamic Frame Generation may enter a state that results in lower instantaneous performance.

Reflex

Q: Should I add GPU hardware, vendor, or driver version check for Reflex?

A: Do NOT do any GPU hardware, vendor, or driver version check. As long as the Reflex plugin is supported, always make all set feature constants and evaluate feature calls to the Reflex Streamline plugin.

- The Reflex Streamline plugin handles all such abstractions. It is completely transparent to the game.
- PCL Stats requires the constants and the markers to properly measure PC latency. It works for all GPU hardware, vendors, and driver versions.
- Reflex Low Latency mode does not work for all GPU hardware, vendors, and drivers versions, but it is already abstracted by the Reflex Streamline plugin.
- The game only needs to disable/enable the Reflex Low Latency UI using the lowLatencyAvailable setting, as detailed below.
- It is important to measure latency, even without Reflex Low Latency.

Q: I don't have any GPU hardware, vendor, or driver version check for the Reflex plugin. The same Reflex calls are made on non-NVIDIA HW. Why is PCL still not working?

A: There needs to be at least 1 set feature constants call made for PCL Stats to initialize and start working.

- Sometimes, without the Reflex UI on non-NV GPUs (because lowLatencyAvailable == false), the game may not call set feature constants at all. That may be what's missing.

Q: Should the Reflex marker calls be skipped when Reflex Low Latency mode is Off?

A: Do NOT skip any Reflex calls when Reflex Low Latency mode is set to Off.

- The markers are also used to measure latency.
- It is important to measure latency, even when Reflex Low Latency mode is Off.

Q: How to check if Reflex Low Latency is available on the host system to disable/enable UI?

A: The game should check lowLatencyAvailable to determine whether to show the Reflex Low Latency UI or not.

- Use lowLatencyAvailable for UI only. **As long as the Reflex Streamline plugin is supported, the game should do everything else Reflex related the same way, even when lowLatencyAvailable is false.**

Q: Reflex sleep is negatively impacting framerate. What is going on?

A: Reflex On should not impact more than 4% FPS. Reflex On + Boost should not impact more than 7% FPS.

- If the impact is slightly higher, then please contact your Producer/CM so they can file a bug and have the driver team investigate further.
- If the impact is significantly higher, please read on.

Does it repro with On + Boost only, or does it repro with On as well? If it is On + Boost only, then it is likely caused by RTBO (Render Thread Bound Optimization).

- Another symptom is that FPS is capped to 30.
- Try setting bUseMarkersToOptimize = false to verify if the issue goes away.
- This can happen if the time within render submit start and render submit end/present end markers increases as Reflex sleep increases.
- The problem can usually be avoided by having better render submit and present start/end markers. But some games may just not be compatible with RTBO and cannot set bUseMarkersTopOptimize = true.

Is FPS capped to 15? If so, it is likely a Reflex controller issue that the driver team should investigate.

Huge FPS impact is likely caused by the game calling Reflex sleep more than once per frame. Reflex sleep should not be called more than once per frame.

- It can happen if the game at some point after Reflex sleep and before simulation starts decides to restart the frame from the beginning. In that case, the Reflex sleep might have been called twice or more, and therefore impact the FPS greatly.
- Without Reflex sleep, restarting the frame from the beginning and repeating the input message and other steps before the check would have no problem. But with Reflex sleep, restarting a frame would call it and sleep multiple times. That's a problem.
- One possible solution in this case is to make sure to not call Reflex sleep more than once before simulation start.

Q: What are the HW/SW requirements for Reflex features?

A: Please refer to the following table:

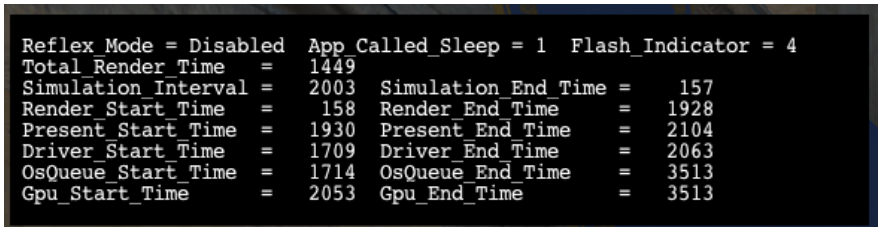
Feature	GPU IHV	GPU HW	Driver Version	Support Check	Key Setting / Marker
Reflex Low Latency	NVIDIA Only	GeForce 900 Series and newer	456.38 or higher	sl::ReflexSettings::lowLatencyAvailable	sl::ReflexConstants::mode
Auto-Configure Reflex Analyzer	NVIDIA Only	GeForce 900 Series and newer	521.60 or higher	sl::ReflexSettings::flashIndicatorDriverControlled	sl::ReflexMarker::eReflexMarkerTriggerFlash
Frame Rate Limiter	NVIDIA Only	All	All	Always	sl::ReflexConstants::frameLimitUs
PC Latency Stats	All	All	All	Always	sl::ReflexMarker::eReflexMarkerPCLatencyPing

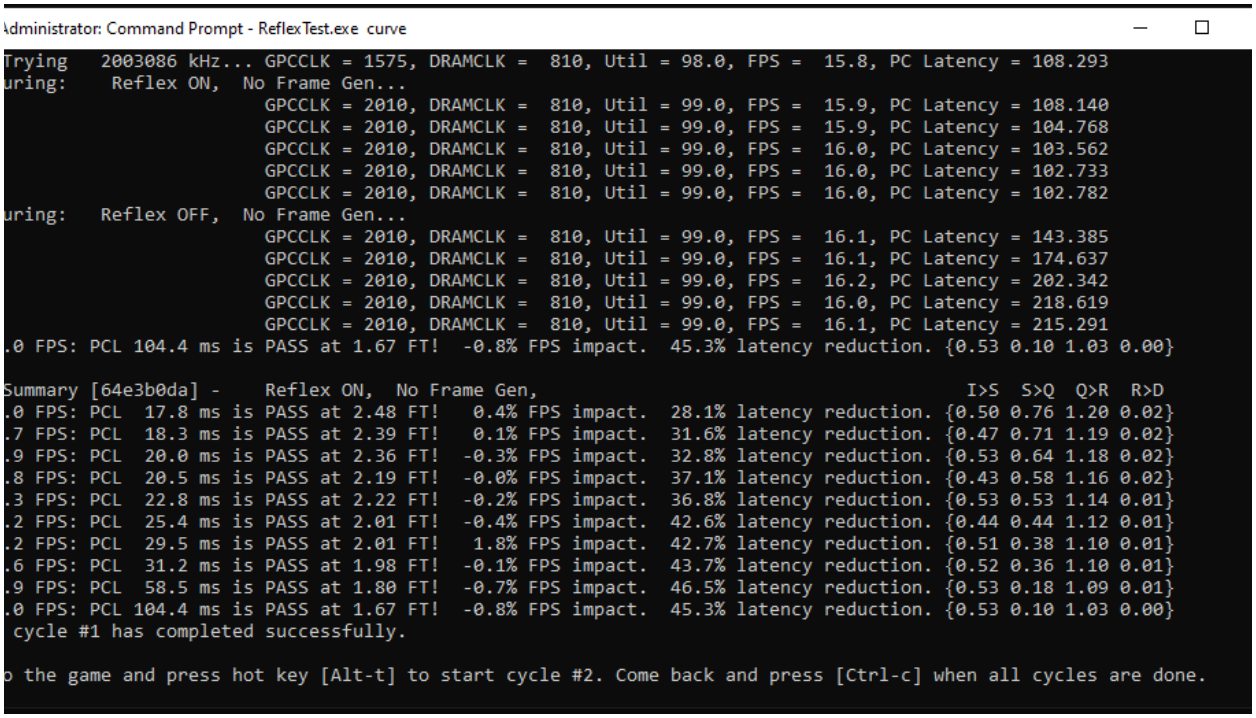
NOTE: The sub-features are distinct to each other without any cross-dependencies. Everything is abstracted within the plugin and is transparent to the application. The application should not explicitly check for GPU HW, vendor, and driver version. The application should do everything the same regardless of sub-feature support and enablement. The only 2 exceptions are Reflex UI and Reflex Flash Indicator (RFI). The application must disable Reflex UI based on sl::ReflexSettings::lowLatencyAvailable. And the application must not send sl::ReflexMarker::eReflexMarkerTriggerFlash when sl::ReflexSettings::flashIndicatorDriverControlled is false.

Q: How can I confirm that Reflex is enabled and working on NVIDIA hardware?

A: You can quickly and easily verify Reflex is enabled and functioning properly by running a simple test found within the NVIDIA Reflex SDK.

1. Browse to “Reflex_Verification” folder
2. Open Command Prompt (run as Administrator) and run FVSKSetup.exe
3. From Command Prompt (run as Administrator), run ReflexTestSetup.bat
4. Start game - you should see the Reflex Verification HUD in-game which will show whether Reflex is enabled
    - Reflex Verification HUD requires driver 531.18 or higher
    - You may need to add the game to the app profile: NVIDIA Control Panel -> 3D Settings -> Program Settings -> Select Program to Customize -> Add -> Select a program -> Add Selected Program -> Apply
5. Press “ALT+T” to start test (you’ll hear 2 beeps)
6. After ~5min, test should be done (you’ll hear 3 beeps)
7. Reference the Command Prompt and ensure “passed”





Q: How can I confirm that Reflex is enabled and working on non-NVIDIA hardware?

A: You can quickly and easily verify Reflex is enabled and functioning properly by running a simple test within the NVIDIA Reflex SDK.

1. Browse to “Reflex_Verification” folder
2. Open Command Prompt (run as Administrator) and run PrintPCL.exe
3. Start game
4. Press “ALT+T” to start test
5. Refer to the command prompt and confirm the PCL value. If the value is not 0.0, then PCL is working.
6. Press “Ctrl + C” to exit test

Q: Why is GPU Utilization being reported as 0%?

A: This issue can arise after you’ve just updated your graphics card drivers or FVSDK. Please be sure to restart the system after the installations are completed before you run the test.

Streamline 2.2.1 New Items - Overview

DLSS Ray Reconstruction

- New Particle Layer API

DLSS Frame Generation

- New Distortion Map API
- Support for Dynamic Frame Generation which can intelligently enable/disable DLSS Frame Generation to help ensure positive performance scaling

Miscellaneous

- Bug fixes and stability enhancements

(***including the following features introduced in Streamline 2.2.0***)

DLSS Ray Reconstruction

- DLSS Ray Reconstruction is a new neural network for all GeForce RTX GPUs that improves the image quality of path-traced and intensive ray-traced content. Ray Reconstruction replaces hand-tuned denoisers with an NVIDIA supercomputer-trained AI network that generates higher-quality pixels in between sampled rays. Plugin “sl.dlss_d”

Auto Scene Change Detection for DLSS Frame Generation

- Discards incorrect frames during fast scene transitions and cuts, leaving only the correct frame, removing the split-second of artifacting that some users noticed. See section 21.0 "Auto Scene Change Detection" in `ProgrammingGuideDLSS_G.md` for more information.

Miscellaneous

- Added support for multiple viewports to the sl.nis plugin
- Added a new structure `SubresourceRange` for specifying Vulkan subresource range information
- Moved SL Feature ID definitions to a central location in `sl.h`
- Fixed a bug where only the first 8 bytes of child-classes of `sl::StructType` would be used for comparison operators

(***including the following features introduced in Streamline 2.1.0***)

OTA UPDATE

- Streamline plugins can now be updated via over-the-air updates

DLAA

- DLAA is now supported as an option for DLSS Super Resolution.

DLSS Frame Generation

- Multiple performance, image quality, and memory optimizations

(***including the following features introduced in Streamline 2.0.0***)

FEATURE ID

- No longer provided as enum but rather as constant expression unsigned integers.
- Core feature IDs declared and defined in sl.h while specific feature IDs are now located in their respective header files.

ENUM NAMING

- All enums have been converted to enum class Name::eValue format.

ERROR REPORTING

- All SL functions now return sl::Result (new header sl_result.h)
- Still required to monitor error logging callbacks to catch every single possible error at the right time but this improves handling of the most common errors.
- Introduced helper macro SL_FAILED.
- Helper method added to convert sl::Result to a string.

VERSIONING

- New API slGetFeatureVersion returns both SL and NGX (if any) versions.

REQUIREMENTS REPORTING

- Removed slGetFeatureConfiguration which was returning JSON since it is not always convenient for eve
- New API slGetFeatureRequirements is added to provide info about OS, driver, HWS, rendering API, VK extensions and other requirements
- Added getVkPhysicalDeviceVulkan12Features and getVkPhysicalDeviceVulkan13Features helper functions in the new sl_helpers_vk.h header.

CONSTANTS VS SETTINGS

- Generic slSetFeatureConsts and slGetFeatureSettings API have been removed and new API slGetFeatureFunction has been added.
- Each SL feature exports a set of functions which are used to perform feature specific task.
- New helper macro SL_FEATURE_FUN_IMPORT and helper functions in the related sl_$feature.h headers.
- Helper functions added to each per-feature header to make importing easy.

IS FEATURE SUPPORTED AND ADAPTER BIT-MASK

- slIsFeatureSupported is modified to use adapter LUID which is easily obtained from DXGIAdapterDesc or VK physical device.
- Engines are already enumerating adapters so this should fit in nicely with the existing code.
- When using VK, the host can provide VkPhysicalDevice instead of LUID if needed.

TRUE FPS

- Removed actualFrameTimeMs and timeBetweenPresentsMs and replaced them with an integer value numFramesActuallyPresented.

IMPROVED VIEWPORT SUPPORT FOR DLSS-FG

- Host can specify any viewport id when calling slDLSSGSetOptions rather than forcing viewport 0 all the time.

"NOT RENDERING GAME FRAMES" FLAG

- Removed completely since it has become redundant.
- Host is now required to turn DLSS-FG on/off using slDLSSGSetOptions

BUFFER COPIES AND TAGGING

- slSetTag API has been expanded to include command list and resource life-cycle information.
- If needed, resources are automatically copied internally by SL.
- slEvaluateFeature can be used to tag resources locally (tags only valid for the specific evaluate call, see programming guide for details)

FRAME ID

- New API slGetNewFrameToken is added to allow host to obtain frame handle for the next frame.
- Same handle is then passed around to all SL calls.
- If host wants to fully control frame counting the frame index can be provided as input

MULTIPLE DEVICES

- New API slSetD3DDevice and slSetVulkanInfo can be used to specify which device SL should be using.

OBTAINING NATIVE INTERFACES

- When using 3rd party libraries (including NVAPI) it is not advisable to pass SL proxies as inputs.
- New API slGetNativeInterface added to allow access to native interfaces as needed.

MANUAL HOOKING

- Removed slGetHook* API

- New API slGetNativeInterface in combination with slUpgradeIntercace is now used to significantly simplify manual hooking.

(***including the following features introduced in Streamline 1.5.x***)

RESOURCE STATE TRACKING

- SL no longer tracks resource states, host is responsible for providing correct states when tagging resources.

COMMAND LIST STATE TRACKING

- SL by default disables any command list (CL) tracking, host is responsible for restoring CL state after each slEvaluateFeature call.

DISABLING OF THE INTERPOSER

- SL disables interposer (DXGI/D3D proxies) unless they are explicitly requested by at least one supported plugin

VERIFYING SL DLLs

- SL now includes sl_security.h header as part of the main SDK (no longer needed to download the SL sample SDK)

OS VERSION DETECTION

- Switched to obtaining the correct version from the kernel32.dll product description instead of using various MS APIs which return different/incorrect versions
- Flag PreferenceFlags::eBypassOSVersionCheck in sl::Preferences can be used to opt-out from the OS detection (off by default and highly discouraged to use)

OTA OPT-IN

- Expanded sl::Preferences to include PreferenceFlags::eAllowOTA to automatically opt-in in the OTA program (SL and NGX). This flag is set by default.

DLSS PRESETS

- Host can change DL networks (presets) for the DLSS by modifying sl::DLSSOptions.