

NVIDIA Virtual GPU Software Management SDK User Guide

Release 21.0

NVIDIA Corporation

Nov 05, 2026

Contents

1 Introduction to the NVIDIA vGPU Software Management SDK 1
1.1 NVIDIA vGPU Software Management Interfaces . . . 1
1.2 Introduction to NVML . . . 3
1.3 NVIDIA vGPU Software Management SDK contents . . . 3

2 Managing vGPUs from a hypervisor by using NVML 5
2.1 Determining whether a GPU supports hosting of vGPUs . . . 5
2.2 Discovering the vGPU capabilities of a physical GPU . . . 5
2.3 Getting the properties of a vGPU type . . . 6
2.4 Getting the properties of a vGPU instance . . . 7
2.5 Building an NVML-enabled application for a vGPU host . . . 10

3 Managing vGPUs from a guest VM 11
3.1 NVIDIA vGPU Software Server Interfaces for GPU Management from a Guest VM . . . 11
3.2 How GPU engine usage is reported . . . 12
3.3 Using NVML to manage vGPUs . . . 12
3.3.1 Determining whether a GPU is a vGPU or pass-through GPU . . . 12
3.3.2 Physical GPU properties that do not apply to a vGPU . . . 12
3.3.2.1 GPU identification properties that do not apply to a vGPU . . . 13
3.3.2.2 `InfoROM` properties that do not apply to a vGPU . . . 13
3.3.2.3 GPU operation mode properties that do not apply to a vGPU . . . 13
3.3.2.4 PCI Express properties that do not apply to a vGPU . . . 14
3.3.2.5 Environmental properties that do not apply to a vGPU . . . 14
3.3.2.6 Power consumption properties that do not apply to a vGPU . . . 14
3.3.2.7 ECC properties that do not apply to a vGPU . . . 15
3.3.2.8 Clocks properties that do not apply to a vGPU . . . 15
3.3.3 Building an NVML-enabled application for a guest VM . . . 15
3.4 Using Windows Performance Counters to monitor GPU performance . . . 16
3.5 Using NVWMI to monitor GPU performance . . . 16
3.5.1 `Gpu` instance properties that do not apply to vGPUs . . . 17
3.5.2 `PcieLink` instance properties that do not apply to vGPUs . . . 17
3.5.3 Enabling and disabling GPM metrics . . . 17
3.5.4 Monitoring GPU performance with GPM metrics . . . 18

4 Notices 21

Chapter 1. Introduction to the NVIDIA vGPU Software Management SDK

The NVIDIA vGPU software Management SDK enables third party applications to monitor and control NVIDIA physical GPUs and virtual GPUs that are running on virtualization hosts. The NVIDIA vGPU software Management SDK supports control and monitoring of GPUs from both the hypervisor host system and from within guest VMs.

NVIDIA vGPU software enables multiple virtual machines (VMs) to have simultaneous, direct access to a single physical GPU, using the same NVIDIA graphics drivers that are deployed on non-virtualized operating systems. For an introduction to NVIDIA vGPU software, see Virtual GPU Software User Guide.

1.1. NVIDIA vGPU Software Management Interfaces

The local management interfaces that are supported within an NVIDIA vGPU software server are shown in Figure 1.1.

For a summary of the NVIDIA vGPU software server interfaces for GPU management, including the hypervisors and guest operating systems that support each interface, and notes about how each interface can be used, see Table 1.1.

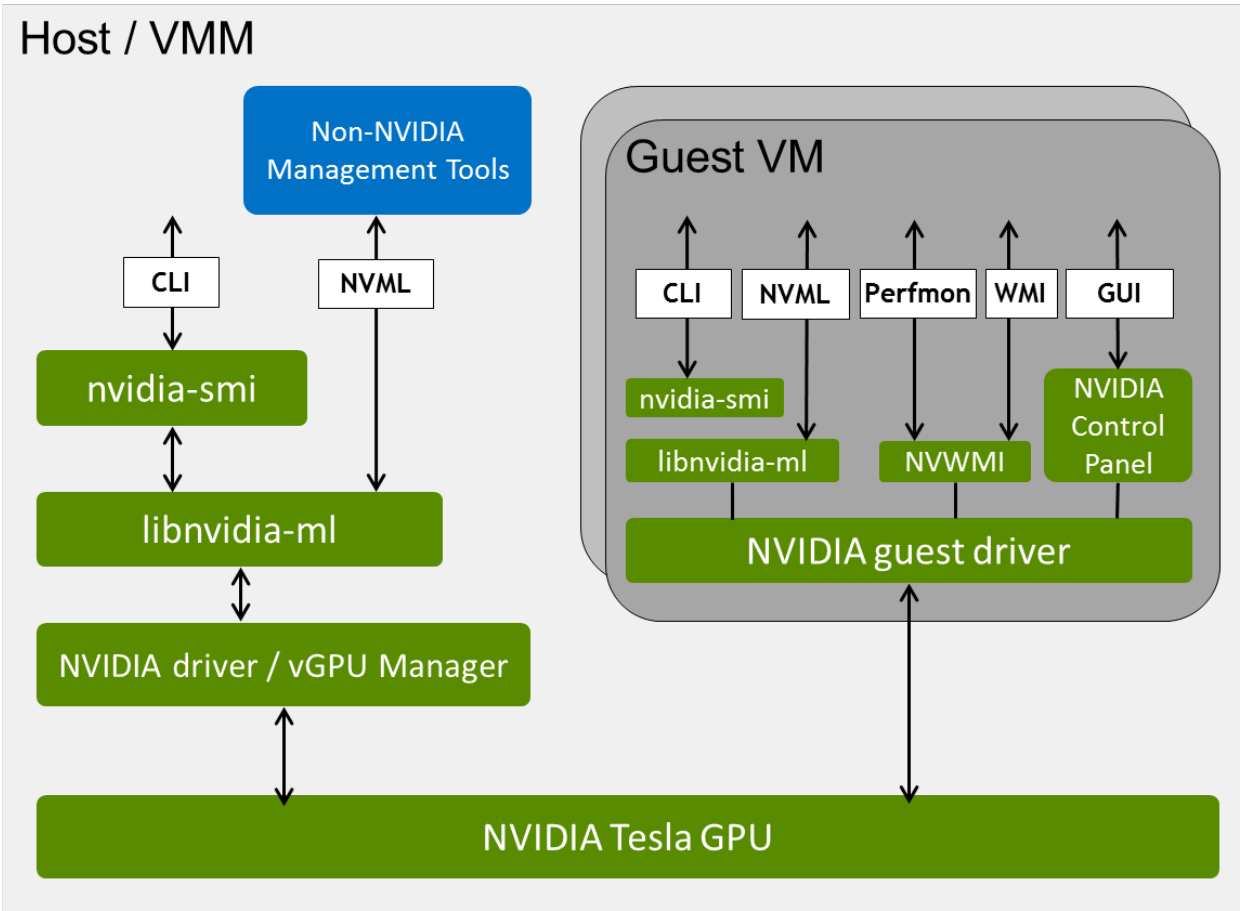


Figure 1.1: NVIDIA vGPU Software server interfaces for GPU management

Table 1.1: Summary of NVIDIA vGPU Software server interfaces for GPU management

Interface	Hypervisor	Guest OS	Notes
`nvidia-smi` command	Any supported hypervisor	Windows, 64-bit Linux	Command line, interactive use
NVIDIA Management Library (NVML)	Any supported hypervisor	Windows, 64-bit Linux	Integration of NVIDIA GPU management with third-party applications
NVIDIA Control Panel	—	Windows	Detailed control of graphics settings, basic configuration reporting
Windows Performance Counters	—	Windows	Performance metrics provided by Windows Performance Counter interfaces
NVWMI	—	Windows	Detailed configuration and performance metrics provided by Windows WMI interfaces

1.2. Introduction to NVML

NVIDIA Management Library (NVML) is a C-based API for monitoring and managing various states of NVIDIA GPU devices. NVML is delivered in the NVIDIA vGPU software Management SDK and as a runtime version:

- The NVIDIA vGPU software Management SDK is distributed as separate archives for Windows and Linux.

  The SDK provides the NVML headers and stub libraries that are required to build third-party NVML applications. It also includes a sample application.

- The runtime version of NVML is distributed with the NVIDIA vGPU software host driver.

Each new version of NVML is backwards compatible, so that applications written to a version of the NVML can expect to run unchanged on future releases of the NVIDIA vGPU software drivers and NVML library.

For details about the NVML API, see:

- NVML API Reference Manual
- NVML man pages

1.3. NVIDIA vGPU Software Management SDK contents

The SDK consists of the NVML developer package and is distributed as separate archives for Windows and Linux:

- Windows: `grid_nvml_sdk_621.10.zip` ZIP archive

- Linux: `grid_nvml_sdk_620.70.04.tgz` GZIP-compressed tar archive

The contents of these archives are summarized in the following table.

Content	Windows Folder	Linux Directory
SDK Samples And Tools License Agreement		
Virtual GPU Software Management SDK User Guide (this document)		
NVML API documentation, on Linux as man pages	`nvml_sdk/doc/`	`nvml_sdk/doc/`
Sample source code and platform-dependent build files: - Windows: Visual C project - Linux: Make file	`nvml_sdk/example/`	`nvml_sdk/examples/`
NVML header file	`nvml_sdk/include/`	`nvml_sdk/include/`
Stub library to allow compilation on platforms without an NVIDIA driver installed	`nvml_sdk/lib/`	`nvml_sdk/lib/`

Chapter 2. Managing vGPUs from a hypervisor by using NVML

NVIDIA vGPU software supports monitoring and control of physical GPUs and virtual GPUs that are running on virtualization hosts. NVML includes functions that are specific to managing vGPUs on NVIDIA vGPU software virtualization hosts. These functions are defined in the `nvml_grid.h` header file.

> **Note**
>
> NVIDIA vGPU software does not support the management of pass-through GPUs from a hypervisor. NVIDIA vGPU software supports the management of pass-through GPUs only from within the guest VM that is using them.

2.1. Determining whether a GPU supports hosting of vGPUs

If called on platforms or GPUs that do not support NVIDIA vGPU, functions that are specific to managing vGPUs return one of the following errors:

- `NVML_ERROR_NOT_SUPPORTED`
- `NVML_ERROR_INVALID_ARGUMENT`

To determine whether a GPU supports hosting of vGPUs, call the `nvmlDeviceGetVirtualizationMode()` function.

A vGPU-capable device reports its virtualization mode as `NVML_GPU_VIRTUALIZATION_MODE_HOST_VGPU`.

2.2. Discovering the vGPU capabilities of a physical GPU

To discover the vGPU capabilities of a physical GPU, call the functions in the following table.

Function	Purpose
`nvmlDeviceGetVirtualizationMode()`	Determine the virtualization mode of a GPU. GPUs capable of hosting virtual GPUs report their virtualization mode as `NVML_GPU_VIRTUALIZATION_MODE_HOST_VGPU`.
`nvmlDeviceGetSupportedVgpus()`	Return a list of vGPU type IDs that are supported by a GPU.
`nvmlDeviceGetCreatableVgpus()`	Return a list of vGPU type IDs that can currently be created on a GPU. The result reflects the number and type of vGPUs that are already running on the GPU. ▶ If no vGPUs are running on the GPU, the list contains the IDs of all vGPU types that the GPU supports. ▶ If one or more vGPUs are running on the GPU, but the GPU is not fully loaded, the list contains only the IDs of the types of the vGPUs that have the same amount of frame buffer as the vGPUs that are already running. ▶ If the GPU is fully loaded, the list contains no vGPU type IDs.
`nvmlDeviceGetActiveVgpus()`	Return a list of handles for vGPUs currently running on a GPU.
`nvmlDeviceGetVgpuMetadata()`	Return a vGPU metadata structure for the physical GPU.
`nvmlDeviceGetVgpuCapabilities()`	Determine whether the requested vGPU capability is supported by the specified physical GPU.

2.3. Getting the properties of a vGPU type

To get the properties of a vGPU type, call the functions in the following table.

Function	Purpose
`nvmlVgpuTypeGetClass()`	Read the class of a vGPU type (for example, Quadro, or NVS)
`nvmlVgpuTypeGetName()`	Read the name of a vGPU type (for example, GRID M60-0Q)
`nvmlVgpuTypeGetDeviceID()`	Read PCI device ID of a vGPU type (vendor/device/subvendor/subsystem)
`nvmlVgpuTypeGetFramebufferSize()`	Read the frame buffer size of a vGPU type
`nvmlVgpuTypeGetNumDisplayHeads()`	Read the number of display heads supported by a vGPU type
`nvmlVgpuTypeGetResolution()`	Read the maximum resolution of a vGPU type's supported display head
`nvmlVgpuTypeGetLicense()`	Read license information required to operate a vGPU type
`nvmlVgpuTypeGetFrameRateLimit()`	Read the static frame limit for a vGPU type
`nvmlVgpuTypeGetMaxInstances()`	Read the maximum number of vGPU instances that can be created on a GPU
`nvmlVgpuTypeGetGpuInstanceProfileId()`	Read the corresponding GPU instance profile ID of a vGPU type. Because NVIDIA vGPU software supports **only** GPUs that do **not** support MIG, this function returns `INVALID_GPU_INSTANCE_PROFILE_ID`.
`nvmlVgpuTypeGetCapabilities()`	Determine whether the vGPU type supports peer-to-peer CUDA transfers over NVLink or GPUDirect® technology.

2.4. Getting the properties of a vGPU instance

To get the properties of a vGPU instance, call the functions in the following table.

Function	Purpose
`nvmlVgpuInstanceGetVmID()`	Read the ID of the VM currently associated with a vGPU instance
`nvmlVgpuInstanceGetUUID()`	Read a vGPU instance's UUID
`nvmlVgpuInstanceGetMdevUUID()`	Read a vGPU instance's virtual function I/O (VFIO) mediated device (`mdev`) UUID (hypervisors based on KVM only)
`nvmlVgpuInstanceGetVmDriverVersion()`	Read the guest driver version currently loaded on a vGPU instance

continues on next page

Table 2.1 – continued from previous page

Function	Purpose
`nvmlVgpuInstanceGetFbUsage()`	Read a vGPU instance's current frame buffer usage
`nvmlVgpuInstanceGetFBCStats()`	Read the following frame buffer capture (FBC) statistics for a vGPU instance: ▶ Count of active FBC sessions ▶ Moving average of new frames captured per second by all active sessions ▶ Moving average of new frame capture latency in microseconds for all active sessions
`nvmlVgpuInstanceGetFBCSessions()`	For each active FBC session on a vGPU instance, read the following statistics: ▶ FBC session ID ▶ Owning PID ▶ vGPU instance identifier ▶ Display ordinal associated with the FBC session. ▶ FBC session type ▶ FBC session flags ▶ Maximum horizontal resolution supported by the session ▶ Maximum vertical resolution supported by the session ▶ Horizontal resolution requested by the caller in the capture call ▶ Vertical resolution requested by the caller in the capture call ▶ Moving average of new frames captured per second by the session ▶ Moving average new frame capture latency in microseconds for the session
`nvmlVgpuInstanceGetLicenseStatus()`	Read a vGPU instance's current license status (licensed or unlicensed)
`nvmlVgpuInstanceGetType()`	Read the vGPU type ID of a vGPU instance
`nvmlVgpuInstanceGetFrameRateLimit()`	Read a vGPU instance's frame rate limit
`nvmlVgpuInstanceGetEncoderStats()`	Read the following encoder statistics for a vGPU instance: ▶ Count of active encoder sessions ▶ One-second trailing average of encoded FPS of all active sessions ▶ One-second trailing average of encode latency in microseconds

continues on next page

Table 2.1 – continued from previous page

Function	Purpose
`nvmlVgpuInstanceGetEncoderSessions()`	For each active encoder session on a vGPU instance, read the following statistics: ▶ Encoder session ID ▶ Owning PID ▶ Codec type, for example, H.264 or H.265 ▶ Encode resolution ▶ One-second trailing averages for encoded FPS and encode latency
`nvmlDeviceGetVgpuUtilization()`	Read a vGPU instance's usage of the following resources as a percentage of the physical GPU's capacity: ▶ 3D/Compute ▶ Frame buffer bandwidth ▶ Video encoder ▶ Video decoder
`nvmlDeviceGetVgpuProcessUtilization()`	For each process running on a vGPU instance, read the process ID and usage by the process of the following resources as a percentage of the physical GPU's capacity: ▶ 3D/Compute ▶ Frame buffer bandwidth ▶ Video encoder ▶ Video decoder
`nvmlDeviceGetGridLicensableFeatures()`	Return a structure containing information about whether a vGPU or a physical GPU supports NVIDIA vGPU software licensing and, if so, additional licensing information such as: ▶ The NVIDIA vGPU software products that can be licensed on the vGPU or physical GPU ▶ License status ▶ The exact name of the NVIDIA vGPU software product for which the vGPU or physical GPU is licensed for and enabled ▶ The year, month, day, hour, minutes, and seconds of the license expiration date in the GMT (UTC+0) timezone.
`nvmlVgpuInstanceGetAccountingMode()`	Read the accounting mode of the vGPU instance
`nvmlVgpuInstanceGetAccountingPids()`	Read the maximum number of processes that can be queried and the current list of process IDs

continues on next page

Table 2.1 – continued from previous page

Function	Purpose
`nvmlVgpuInstanceGetAccountingStats()`	For each process ID returned by `nvmlVgpuInstanceGetAccountingPids()`, read the following statistics: ▶ GPU utilization ▶ Memory utilization ▶ Maximum memory usage ▶ Amount of time the graphics or compute context was active ▶ Start time of the process ▶ Whether the process is still running
`nvmlGetVgpuCompatibility()`	Return compatibility information about a vGPU and a physical GPU, such as: ▶ The VM states from which the vGPU can be run on the physical GPU ▶ Any factors limiting compatibility between the vGPU and the physical GPU
`nvmlVgpuInstanceGetMetadata()`	Return a vGPU metadata structure for a vGPU and its associated VM
`nvmlGetVgpuDriverCapabilities()`	Determine whether the requested vGPU driver capability is supported by the Virtual GPU manager.

2.5. Building an NVML-enabled application for a vGPU host

Functions that are specific to vGPUs are defined in the header file `nvml_grid.h`.

To build an NVML-enabled application for a vGPU host, ensure that you include `nvml_grid.h` in addition to `nvml.h`:

```
#include <nvml.h>
#include <nvml_grid.h>
```

For more information, refer to the sample code that is included in the SDK.

Chapter 3. Managing vGPUs from a guest VM

NVIDIA vGPU software supports monitoring and control within a guest VM of vGPUs or pass-through GPUs that are assigned to the VM. The scope of management interfaces and tools used within a guest VM is limited to the guest VM within which they are used. They cannot monitor any other GPUs in the virtualization platform.

For monitoring from a guest VM, certain properties do not apply to vGPUs. The values that the NVIDIA vGPU software management interfaces report for these properties indicate that the properties do not apply to a vGPU.

3.1. NVIDIA vGPU Software Server Interfaces for GPU Management from a Guest VM

The NVIDIA vGPU software server interfaces that are available for GPU management from a guest VM depend on the guest operating system that is running in the VM.

Interface	Guest OS	Notes
`nvidia-smi` command	Windows, 64-bit Linux	Command line, interactive use
NVIDIA Management Library (NVML)	Windows, 64-bit Linux	Integration of NVIDIA GPU management with third-party applications
NVIDIA Control Panel	Windows	Detailed control of graphics settings, basic configuration reporting
Windows Performance Counters	Windows	Performance metrics provided by Windows Performance Counter interfaces
NVWMI	Windows	Detailed configuration and performance metrics provided by Windows WMI interfaces

3.2. How GPU engine usage is reported

Usage of GPU engines is reported for vGPUs as a percentage of the vGPU's maximum possible capacity on each engine. The GPU engines are as follows:

- Graphics/SM
- Memory controller
- Video encoder
- Video decoder

The amount of a physical engine's capacity that a vGPU is permitted to occupy depends on the scheduler under which the GPU is operating:

- NVIDIA vGPUs operating under the **Best Effort Scheduler** and the **Equal Share Scheduler** are permitted to occupy the full capacity of each physical engine if no other vGPUs are contending for the same engine. Therefore, if a vGPU occupies 20% of the entire graphics engine in a particular sampling period, its graphics usage as reported inside the VM is 20%.
- NVIDIA vGPUs operating under the **Fixed Share Scheduler** can occupy no more than their allocated share of the graphics engine. Therefore, if a vGPU has a fixed allocation of 25% of the graphics engine, and it occupies 25% of the engine in a particular sampling period, its graphics usage as reported inside the VM is 100%.

3.3. Using NVML to manage vGPUs

NVIDIA vGPU software supports monitoring and control within a guest VM by using NVML.

3.3.1. Determining whether a GPU is a vGPU or pass-through GPU

NVIDIA vGPUs are presented in guest VM management interfaces in the same fashion as pass-through GPUs.

To determine whether a GPU device in a guest VM is a vGPU or a pass-through GPU, call the NVML function `nvmlDeviceGetVirtualizationMode()`.

A GPU reports its virtualization mode as follows:

- A GPU operating in pass-through mode reports its virtualization mode as `NVML_GPU_VIRTUALIZATION_MODE_PASSTHROUGH`.
- A vGPU reports its virtualization mode as `NVML_GPU_VIRTUALIZATION_MODE_VGPU`.

3.3.2. Physical GPU properties that do not apply to a vGPU

Properties and metrics other than GPU engine usage are reported for a vGPU in a similar way to how the same properties and metrics are reported for a physical GPU. However, some properties do not apply to vGPUs. The NVML device query functions for getting these properties return a value that indicates that the properties do not apply to a vGPU. For details of NVML device query functions, see Device Queries in *NVML API Reference Manual*.

3.3.2.1 GPU identification properties that do not apply to a vGPU

GPU Property	NVML Device Query Function	NVML return code on vGPU
Serial Number	`nvmlDeviceGetSerial()` vGPUs are not assigned serial numbers.	`NOT_SUPPORTED`
GPU UUID	`nvmlDeviceGetUUID()` vGPUs are allocated random UUIDs.	`SUCCESS`
VBIOS Version	`nvmlDeviceGetVbiosVersion()` vGPU VBIOS version is hard-wired to zero.	`SUCCESS`
GPU Part Number	`nvmlDeviceGetBoardPartNumber()`	`NOT_SUPPORTED`

3.3.2.2 `InfoROM` properties that do not apply to a vGPU

The `InfoROM` object is not exposed on vGPUs. All the functions in the following table return `NOT_SUPPORTED`.

GPU Property	NVML Device Query Function
Image Version	`nvmlDeviceGetInforomImageVersion()`
OEM Object	`nvmlDeviceGetInforomVersion()`
ECC Object	`nvmlDeviceGetInforomVersion()`
Power Management Object	`nvmlDeviceGetInforomVersion()`

3.3.2.3 GPU operation mode properties that do not apply to a vGPU

GPU Property	NVML Device Query Function	NVML return code on vGPU
GPU Operation Mode (Current)	`nvmlDeviceGetGpuOperationMode()` Tesla GPU operating modes are not supported on vGPUs.	`NOT_SUPPORTED`
GPU Operation Mode (Pending)	`nvmlDeviceGetGpuOperationMode()` Tesla GPU operating modes are not supported on vGPUs.	`NOT_SUPPORTED`
Compute Mode	`nvmlDeviceGetComputeMode()` A vGPU always returns `NVML_COMPUTEMODE_PROHIBITED`.	`SUCCESS`
Driver Model	`nvmlDeviceGetDriverModel()` A vGPU supports WDDM mode only in Windows VMs.	SUCCESS (Windows)

3.3.2.4 PCI Express properties that do not apply to a vGPU

PCI Express characteristics are not exposed on vGPUs. All the functions in the following table return `NOT_SUPPORTED`.

GPU Property	NVML Device Query Function
Generation Max	`nvmlDeviceGetMaxPcieLinkGeneration()`
Generation Current	`nvmlDeviceGetCurrPcieLinkGeneration()`
Link Width Max	`nvmlDeviceGetMaxPcieLinkWidth()`
Link Width Current	`nvmlDeviceGetCurrPcieLinkWidth()`
Bridge Chip Type	`nvmlDeviceGetBridgeChipInfo()`
Bridge Chip Firmware	`nvmlDeviceGetBridgeChipInfo()`
Replays	`nvmlDeviceGetPcieReplayCounter()`
TX Throughput	`nvmlDeviceGetPcieThroughput()`
RX Throughput	`nvmlDeviceGetPcieThroughput()`

3.3.2.5 Environmental properties that do not apply to a vGPU

All the functions in the following table return `NOT_SUPPORTED`.

GPU Property	NVML Device Query Function
Fan Speed	`nvmlDeviceGetFanSpeed()`
Clocks Throttle Reasons	`nvmlDeviceGetSupportedClocksThrottleReasons()` `nvmlDeviceGetCurrentClocksThrottleReasons()`
Current Temperature	`nvmlDeviceGetTemperature()` `nvmlDeviceGetTemperatureThreshold()`
Shutdown Temperature	`nvmlDeviceGetTemperature()` `nvmlDeviceGetTemperatureThreshold()`
Slowdown Temperature	`nvmlDeviceGetTemperature()` `nvmlDeviceGetTemperatureThreshold()`

3.3.2.6 Power consumption properties that do not apply to a vGPU

vGPUs do not expose physical power consumption of the underlying GPU. All the functions in the following table return `NOT_SUPPORTED`.

GPU Property	NVML Device Query Function
Management Mode	`nvmlDeviceGetPowerManagementMode()`
Draw	`nvmlDeviceGetPowerUsage()`
Limit	`nvmlDeviceGetPowerManagementLimit()`
Default Limit	`nvmlDeviceGetPowerManagementDefaultLimit()`
Enforced Limit	`nvmlDeviceGetEnforcedPowerLimit()`
Min Limit	`nvmlDeviceGetPowerManagementLimitConstraints()`
Max Limit	`nvmlDeviceGetPowerManagementLimitConstraints()`

3.3.2.7 ECC properties that do not apply to a vGPU

Error-correcting code (ECC) is not supported on vGPUs. All the functions in the following table return `NOT_SUPPORTED`.

GPU Property	NVML Device Query Function
Mode	`nvmlDeviceGetEccMode()`
Error Counts	`nvmlDeviceGetMemoryErrorCounter()` `nvmlDeviceGetTotalEccErrors()`
Retired Pages	`nvmlDeviceGetRetiredPages()` `nvmlDeviceGetRetiredPagesPendingStatus()`

3.3.2.8 Clocks properties that do not apply to a vGPU

All the functions in the following table return `NOT_SUPPORTED`.

GPU Property	NVML Device Query Function
Application Clocks	`nvmlDeviceGetApplicationsClock()`
Default Application Clocks	`nvmlDeviceGetDefaultApplicationsClock()`
Max Clocks	`nvmlDeviceGetMaxClockInfo()`
Policy: Auto Boost	`nvmlDeviceGetAutoBoostedClocksEnabled()`
Policy: Auto Boost Default	`nvmlDeviceGetAutoBoostedClocksEnabled()`

3.3.3. Building an NVML-enabled application for a guest VM

To build an NVML-enabled application, refer to the sample code included in the SDK.

3.4. Using Windows Performance Counters to monitor GPU performance

In Windows VMs, GPU metrics are available as Windows Performance Counters through the `NVIDIA GPU` object.

For access to Windows Performance Counters through programming interfaces, refer to the performance counter sample code included with the NVIDIA Windows Management Instrumentation SDK.

On vGPUs, the following GPU performance counters read as 0 because they are not applicable to vGPUs:

- % Bus Usage
- % Cooler rate
- Core Clock MHz
- Fan Speed
- Memory Clock MHz
- PCI-E current speed to GPU Mbps
- PCI-E current width to GPU
- PCI-E downstream width to GPU
- Power Consumption mW
- Temperature C

3.5. Using NVWMI to monitor GPU performance

In Windows VMs, Windows Management Instrumentation (WMI) exposes GPU metrics in the `ROOT\CIMV2\NV` namespace through NVWMI. NVWMI is included with the NVIDIA driver package. The NVWMI API Reference in Windows Help format is available for download from the NVIDIA website.

For access to NVWMI through programming interfaces, use the NVWMI SDK. The NVWMI SDK, with white papers and sample programs, is included in the NVIDIA Windows Management Instrumentation SDK.

On vGPUs, some instance properties of the following classes do not apply to vGPUs:

- `Gpu`
- `PcieLink`

3.5.1. Gpu instance properties that do not apply to vGPUs

Gpu Instance Property	Value reported on vGPU
gpuCoreClockCurrent	-1
memoryClockCurrent	-1
pciDownstreamWidth	0
pcieGpu.curGen	0
pcieGpu.curSpeed	0
pcieGpu.curWidth	0
pcieGpu.maxGen	1
pcieGpu.maxSpeed	2500
pcieGpu.maxWidth	0
power	-1
powerSampleCount	-1
powerSamplingPeriod	-1
verVBIOS.orderedValue	0
verVBIOS.strValue	—
verVBIOS.value	0

3.5.2. PcieLink instance properties that do not apply to vGPUs

No instances of PcieLink are reported for vGPU.

3.5.3. Enabling and disabling GPM metrics

GPU performance monitoring (GPM) metrics are supported on all Windows guest OS releases that are supported by NVIDIA vGPU software. GPM metrics are supported only on MIG-backed vGPUs that occupy an entire GPU instance on GPUs based on the NVIDIA Blackwell architecture or a later architecture.

GPM metrics are disabled by default. To monitor GPU performance with GPM metrics, you must first enable them.

This task requires administrator privileges.

Perform this task in an elevated Windows PowerShell window (**Run as administrator**) in the guest VM.

Before you begin, ensure that the NVWMI service is running.

- To enable GPM metrics, type the following command:

```
Invoke-CimMethod -Namespace 'root\cimv2\nv' -ClassName System -MethodName
↪enableGpmPerfCounter
```

GPM metrics do **not** remain enabled after the NVWMI service or the guest VM is restarted. After the NVWMI service or the VM is restarted, enable GPM metrics again.

- To disable GPM metrics, type the following command:

```
Invoke-CimMethod -Namespace 'root\cimv2\nv' -ClassName System -MethodName
↪disableGpmPerfCounter
```

3.5.4. Monitoring GPU performance with GPM metrics

GPM metrics are supported only on MIG-backed vGPUs that occupy an entire GPU instance on GPUs based on the NVIDIA Blackwell architecture or a later architecture.

Before you begin, ensure that the following prerequisites are met:

- The GPU is in MIG mode.
- The GPU supports GPM.
- GPM metrics are enabled as explained in *Enabling and disabling GPM metrics*.

If any of these prerequisites is **not** met, any attempt to monitor GPU performance with GPM metrics returns no data.

The following table maps each GPM metric's WMI property to its Performance Monitor counter.

WMI property	Performance Monitor counter
`graphicsUtil`	% Graphics Util
`smUtil`	% SM Util
`smOccupancy`	% SM Occupancy
`integerUtil`	% Integer Util
`anyTensorUtil`	% Any Tensor Util
`dfmaTensorUtil`	% DFMA Tensor Util
`hmmaTensorUtil`	% HMMA Tensor Util
`immaTensorUtil`	% IMMA Tensor Util
`dramBwUtil`	% DRAM BW Util
`fp64Util`	% FP64 Util
`fp32Util`	% FP32 Util
`fp16Util`	% FP16 Util
`nvdec0Util`- `nvdec7Util`	% NVDEC 0 Util-% NVDEC 7 Util
`nvjpg0Util`- `nvjpg7Util`	% NVJPG 0 Util-% NVJPG 7 Util
`nvofa0Util`- `nvofa1Util`	% NVOFA 0 Util-% NVOFA 1 Util
`nvenc0Util`- `nvenc3Util`	% NVENC 0 Util-% NVENC 3 Util

You can monitor GPU performance with GPM metrics by using either of the following methods:

- Using any WMI client to query the `GpmMetrics` class for GPM metrics

  For example, you can use the following command from a PowerShell prompt in a Windows guest VM.

  ```
  Get-CimInstance -Namespace 'root\cimv2\nv' -ClassName GpmMetrics
  ```

  The following example shows the query results. In the query results, `instanceCount` is `1`. The example omits the remaining metric properties, which are listed in the preceding table.

  ```
  id                : 1
  instanceCount     : 1
  gpuInstanceId     : {0}
  computeInstanceId : {0}
  smUtil            : {0}
  smOccupancy       : {0}
  ...
  ```

- Using any application that reads performance counters to read the `NVIDIA GPU GPM` counters

  For example, you can use Windows Performance Monitor to view the counters under the `NVIDIA GPU GPM` object.

  The object lists one instance, named in the following form:

  ```
  GPU<index> GI<id> CI<id> (GPU Id : <id>)
  ```

  Selecting that instance makes the GPM counters available for monitoring.

  The following example shows the `NVIDIA GPU GPM` object in the **Add Counters** dialog box.

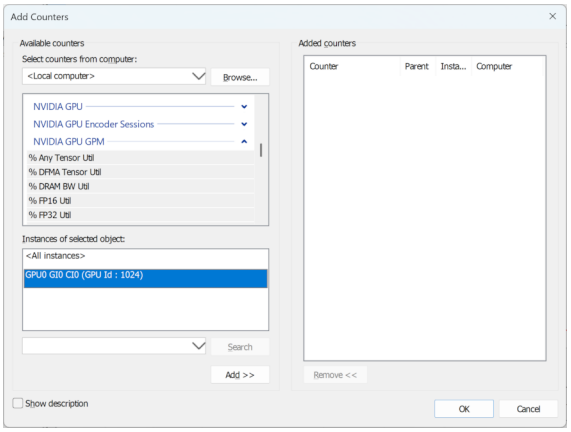


Figure 3.1: `NVIDIA GPU GPM` object in the Add Counters dialog box

The first time you query or view GPM metrics after they are enabled, all metrics are 0 because they are derived from a pair of samples. All metrics for an idle MIG instance are also 0.

Chapter 4. Notices

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a

third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA, the NVIDIA logo, NVIDIA GRID, NVIDIA GRID vGPU, NVIDIA Maxwell, NVIDIA Pascal, NVIDIA Turing, NVIDIA Volta, GPUDirect, Quadro, and Tesla are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

©2013-2026, NVIDIA Corporation