NVML

vR550 | March 2024

Reference Manual

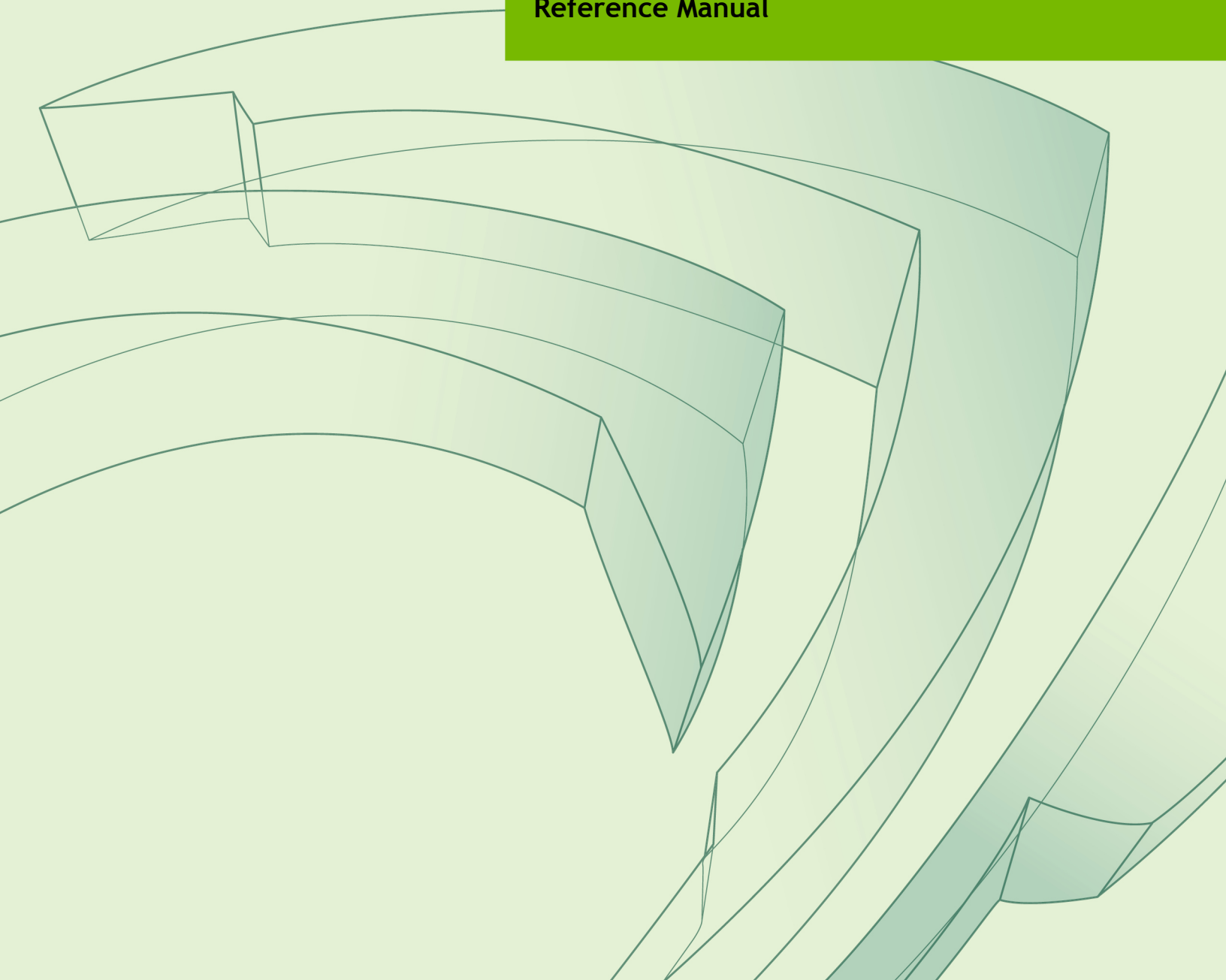

TABLE OF CONTENTS

Chapter 1. NVML API REFERENCE

The NVIDIA Management Library (*NVML*) is a C-based programmatic interface for monitoring and managing various states within NVIDIA Tesla™ GPUs. It is intended to be a platform for building 3rd party applications, and is also the underlying library for the NVIDIA-supported **`nvidia-smi`** tool. NVML is thread-safe so it is safe to make simultaneous NVML calls from multiple threads.

API Documentation

Supported OS platforms:

- Windows 64-bit: Windows Server 2019, Windows Server 2016, Windows Server 2012 R2, Windows 10
- Linux: 64-bit

Supported products:

- Full Support
  - NVIDIA Tesla Line:
    - A100, A40, A30, A16, A10
    - H100
    - T4
    - V100
    - P100, P40, P4, P6
    - M60, M40, M6, M4
    - K80, K520
  - NVIDIA Quadro Line:
    - RTX 8000, RTX 6000, RTX 5000, RTX 4000, RTX 3000
    - GV100, GP100, P6000, P5200, P5000, P4000, P2200, P2000, P1000, P620, P600, P400
    - M6000 24GB, M6000, M5000, M4000, M2000

        - K6000, K5200, K5000, K4000, K4200, K2200, K2000, K2000D, K1200, K620, K600, K420, 410
    - NVIDIA GeForce Line:
        - none
- Limited Support
    - NVIDIA Tesla Line: All other current and previous generation Tesla-branded parts
    - NVIDIA Quadro Line: All other current and previous generation Quadro-branded parts
    - NVIDIA GeForce Line: All current and previous generation GeForce-branded parts

The NVML library can be found at the following locations on Windows

- Standard driver install: **`%ProgramW6432%\"NVIDIA Corporation"\NVSMI\`**
- DCH driver install: **`\Windows\System32`**

Note that these libraries will not be added to the path on Windows. To dynamically link to NVML, add this path to the PATH environmental variable. To dynamically load NVML, call **`LoadLibrary`** with this path.

On Linux the NVML library is named "**`libnvidia-ml.so`**" and can be found on the standard library path. To link against the NVML library add the **`-lnvidia-ml`** flag to your linker command.

The NVML API is divided into five categories:

- Support Methods:
    - Initialization and Cleanup
- Query Methods:
    - System Queries
    - Device Queries
    - Unit Queries
- Control Methods:
    - Unit Commands
    - Device Commands
- Event Handling Methods:
    - Event Handling Methods
- Error reporting Methods
    - Error Reporting

Chapter 2. KNOWN ISSUES

This is a list of known NVML issues in the current driver:

- On systems where GPUs are NUMA nodes, the accuracy of FB memory utilization provided by NVML depends on the memory accounting of the operating system. This is because FB memory is managed by the operating system instead of the NVIDIA GPU driver. Typically, pages allocated from FB memory are not released even after the process terminates to enhance performance. In scenarios where the operating system is under memory pressure, it may resort to utilizing FB memory. Such actions can result in discrepancies in the accuracy of memory reporting.
- On Linux, GPU Reset can't be triggered when there is pending GPU Operation Mode (GOM) change.
- On Linux, GPU Reset may not successfully change pending ECC mode. A full reboot may be required to enable the mode change.
- **`nvmlAccountingStats`** supports only one process per GPU at a time (CUDA proxy server counts as one process).
- **`nvmlAccountingStats_t.time`** reports time and utilization values starting from cuInit till process termination. Future driver versions might change this behavior slightly and account process only from **`cuCtxCreate`** till **`cuCtxDestroy`**.
- On GPUs from Fermi family, current P0 clocks (reported by **`nvmlDeviceGetClockInfo`**) can differ from max clocks by a few MHz.

Chapter 3. CHANGE LOG

This chapter list changes in API and bug fixes that were introduced to the library.

Changes between v545 and v550

The following new functionality is exposed on NVIDIA display drivers version 550 Production or later.

- Added nvmlDeviceGetNumaNodeId to query the NUMA node of a GPU.
- Added new GPM metric ID **`NVML_GPM_METRIC_NVOFA_1_UTIL`** to nvmlGpmMetricId_t.
- Added new field ID **`NVML_FI_DEV_IS_MIG_MODE_INDEPENDENT_MIG_QUERY_CAPABLE`**, to check MIG query capable device irrespective of MIG mode.
- Deprecated **`NVML_P2P_CAPS_INDEX_PROP`** and added **`NVML_P2P_CAPS_INDEX_PCI`** to reflect the same P2P capability.
- Added **`nvmlDeviceGetProcessesUtilizationInfo`** to retrieve the recent utilization and process ID for all running processes.
- Added new **`struct nvmlProcessesUtilizationInfo_t`**, which includes the new utilization of NVJPG and NVOFA.
- Added **`nvmlDeviceGetVgpuInstancesUtilizationInfo`** to retrieve the recent utilization for vGPU instances running on a physical GPU.
- Added **`nvmlDeviceGetVgpuProcessesUtilizationInfo`** to retrieve the recent utilization for processes running on vGPU instances on a physical GPU.
- Added **`nvmlDeviceSetVgpuHeterogeneousMode`** to enable or disable vGPU heterogenous mode for the device.
- Added **`nvmlDeviceGetVgpuHeterogeneousMode`** to query the vGPU heterogenous mode for the device.
- Added **`nvmlVgpuInstanceGetPlacementId`** to query placement ID of the active vGPU instance.

- Added **nvmlDeviceGetVgpuTypeSupportedPlacements** to query the supported vGPU placement IDs of a vGPU type.
- Added **nvmlDeviceGetVgpuTypeCreatablePlacements** to query the creatable vGPU placement IDs of a vGPU type.
- Added support to display confidential compute protected memory along with fb and bar1 in nvidia-smi pmon and dmon commands.
- Added **nvmlDeviceGetGpuFabricInfoV** to query GPU Fabric Probe Info for the device.
- Deprecated **nvmlDeviceGetGpuFabricInfo**. This function should not be used, and will be removed in a future release. Use **nvmlDeviceGetGpuFabricInfoV** instead.
- Modified **nvmlDeviceGetGpuInstanceProfileInfo** and **nvmlDeviceGetGpuInstancePossiblePlacements** to no longer require MIG being enabled.
- Added new encoder type **NVML_ENCODER_QUERY_AV1** and **NVML_ENCODER_QUERY_UNKNOWN** to enumeration **nvmlEncoderType_t**.xxxxxxxxxxxxxxx
- Added **nvmlSystemSetConfComputeKeyRotationThresholdInfo** to set confidential compute key rotation threshold.
- Added **nvmlSystemGetConfComputeKeyRotationThresholdInfo** to query confidential compute key rotation threshold detail.
- Added **nvmlDeviceSetVgpuCapabilities** to set the desirable vGPU capability of a device.

Changes between v535 and v545

The following new functionality is exposed on NVIDIA display drivers version 545 Production or later.

- Added a new error code **NVML_ERROR_GPU_NOT_FOUND** to be returned if no supported GPUS are found during initialization.
- In **nvmlGpuFabricInfo_t**, **partitionId** has been renamed to **cliqueId**.
- Added new versioned structs **nvmlGpuInstanceProfileInfo_v3_t** and **nvmlComputeInstanceProfileInfo_v3_t**.
- Added **nvmlDeviceGetLastBBXFlushTime** for retrieving the timestamp and duration of the latest flush of the BBX object to the inforom storage.
- Added **NVML_POWER_SCOPE_MEMORY** to report out power usage for GPU Memory.
- Added **nvmlDeviceGetPciInfoExt** which expands **nvmlDeviceGetPciInfo** to also report PCI base and sub classcodes.
- Added new struct **nvmlPciInfoExt_t**, which is used in **nvmlDeviceGetPciInfoExt**.
- Added **nvmlDeviceGetRunningProcessDetailList** API to get information about Compute, Graphics or MPS-Compute processes running on a GPU with protected memory usage info.

Changes between v530 and v535

The following new functionality is exposed on NVIDIA display drivers version 535 Production or later.

- Added **`nvmlDeviceGetSramEccErrorStatus`** to query SRAM ECC error status for the device.
- Added nvmlDeviceGetModuleId for getting device module id.
- Updated nvmlDeviceGetPowerSource API to report undersized power source.
- AddednvmlDeviceGetJpgUtilization and nvmlDeviceGetOfaUtilization APIs.
- Added nvmlSystemGetNvlinkBwMode and nvmlSystemSetNvlinkBwMode APIs.
- Added nvmlDeviceSetVgpuSchedulerState to set the vGPU scheduler state.
- Added new field Id NVML_FI_DEV_IS_RESETLESS_MIG_SUPPORTED for device's resetless MIG capability.
- Added nvmlDeviceGetComputeRunningProcesses_v3 to get information about Compute processes running on a GPU.
- Added nvmlDeviceGetGraphicsRunningProcesses_v3 to get information about Graphics processes running on a GPU.
- Added nvmlDeviceGetMPSComputeRunningProcesses_v3 to get information about MPS-Compute processes running on a GPU.
- Added nvmlDeviceGetRunningProcessDetailList to get information about Compute, Graphics or MPS-Compute processes running on a GPU with protected memory usage info.
- Added **`nvmlDeviceGetLastBBXFlushTime`** for retrieving the timestamp and duration of the latest flush of the BBX object to the inforom storage.
- Added new field Id NVML_FI_DEV_PCIE_COUNT_CORRECTABLE_ERRORS for PCIe correctable errors counter.
- Added new field Id NVML_FI_DEV_PCIE_COUNT_NAKS_RECEIVED for PCIe NAK Receive counter.
- Added new field Id NVML_FI_DEV_PCIE_COUNT_RECEIVER_ERROR for PCIe receiver error counter.
- Added new field Id NVML_FI_DEV_PCIE_COUNT_BAD_TLP for PCIe bad TLP counter.
- Added new field Id NVML_FI_DEV_PCIE_COUNT_NAKS_SENT for NAK Send counter.
- Added new field Id NVML_FI_DEV_PCIE_COUNT_BAD_DLLP for PCIe bad DLLP counter.
- Added new field Id NVML_FI_DEV_PCIE_COUNT_NON_FATAL_ERROR for PCIe non fatal error counter.
- Added new field Id NVML_FI_DEV_PCIE_COUNT_FATAL_ERROR for PCIe fatal error counter.

- Added new field Id NVML_FI_DEV_PCIE_COUNT_UNSUPPORTED_REQ for PCIe unsupported request counter.
- Added new field Id NVML_FI_DEV_PCIE_COUNT_LCRC_ERROR for PCIe LCRC error counter.
- Added new field Id NVML_FI_DEV_PCIE_COUNT_LANE_ERROR for per lane error counter with scope as PCIe lane number.
- Added **nvmlDeviceGetPowerLimits** for retrieving current Power Limits.
- Added **nvmlDeviceGetPowerUsage_v2** to retrieve current power usage.
- Added **nvmlDeviceGetTotalEnergyConsumption_v2** to get current energy consumption.
- Added nvmlDeviceSetPowerManagementLimit_v2 to set the power limit.
- Added new field IDs, **NVML_FI_GPU_POWER_AVERAGE** and **NVML_FI_GPU_POWER_INSTANT**, to query power usage.
- Renamed **nvmlDeviceCcuGetStreamState** to **nvmlGpmQueryIfStreamingEnabled** and **nvmlDeviceCcuSetStreamState** to **nvmlGpmSetStreamingEnabled**.
- Added support to display confidential compute protected memory along with fb and bar1 in **nvidia-smi pmon** and **dmon** commands.
- Added new field IDs **NVML_FI_DEV_TEMPERATURE_SHUTDOWN_TLIMIT**, **NVML_FI_DEV_TEMPERATURE_SLOWDOWN_TLIMIT**, **NVML_FI_DEV_TEMPERATURE_MEM_MAX_TLIMIT**, and **NVML_FI_DEV_TEMPERATURE_GPU_MAX_TLIMIT** to query temperature thresholds on Ada and later architectures.
- Introduced **ClockEventReasons** and related APIs which should be used instead of **ClockThrottleReasons**. Deprecated **ClockThrottleReasons**.
- Added ability to get GPS Temperature Threshold with **nvmlDeviceGetTemperatureThreshold** using the new enum **NVML_TEMPERATURE_THRESHOLD_GPS_CURR**.

Changes between v525 and v530

The following new functionality is exposed on NVIDIA display drivers version 530 Production or later.

- Fixed a typo in **nvmlGpuP2PStatus_t**: added a new enum entry for **NVML_P2P_STATUS_CHIPSET_NOT_SUPPORTED** with the same numeric value as the existing erroneous entry ("**NVML_P2P_STATUS_CHIPSET_NOT_SUPPORED**").
- Added nvmlDeviceGetVgpuSchedulerLog to fetch the vGPU software scheduler logs.
- Added nvmlDeviceGetVgpuSchedulerState to fetch the vGPU software scheduler state.
- Added nvmlDeviceGetVgpuSchedulerCapabilities to fetch the vGPU software scheduler capabilities.

Changes between v520 and v525

The following new functionality is exposed on NVIDIA display drivers version 525 Production or later.

- Added **`nvmlDeviceGetPcieAtomicCaps`** to report PCIe atomic capabilities.
- Added **`nvmlDeviceCcuGetStreamState`** API to report the counter collection unit stream state.
- Added **`nvmlDeviceCcuSetStreamState`** API to set the counter collection unit stream state.
- Removed support for **`NVML_FI_DEV_LINK_SPEED_MBPS_L{0..}`** field Ids in Hopper. Replaced with **`NVML_FI_DEV_NVLINK_GET_SPEED`** with scope as link Id.
- Removed support for **`NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT{0..}`** field Ids in Hopper. Replaced with **`NVML_FI_DEV_NVLINK_ERROR_DL_CRC`** with scope as link Id.
- Removed support for **`NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L{0..}`** field Ids in Hopper. Replaced with **`NVML_FI_DEV_NVLINK_ERROR_DL_REPLAY`** with scope as link Id.
- Removed support for **`NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_{0..}`** field Ids in Hopper. Replaced with **`NVML_FI_DEV_NVLINK_ERROR_DL_RECOVERY`** with scope as link Id.
- Added new field ID NVML_FI_DEV_NVLINK_GET_STATE to get nvlink state.
- Added new field ID NVML_FI_DEV_NVLINK_GET_VERSION to get nvlink version.
- Added new field ID NVML_FI_DEV_C2C_LINK_COUNT to get C2C link count.
- Added new field ID **`NVML_FI_DEV_C2C_LINK_GET_STATUS`** to get C2C link status.
- Added new field ID NVML_FI_DEV_C2C_LINK_GET_MAX_BW to get C2C link bandwidth.

Changes between v515 and v520

The following new functionality is exposed on NVIDIA display drivers version 520 Production or later.

- Added nvmlDeviceGetMemClkVfOffset API to report the MemClk VF offset value.
- Added nvmlDeviceSetMemClkVfOffset API to set the MemClk VF offset value.
- Added nvmlDeviceGetMemClkMinMaxVfOffset API to report the Memory clock min and max VF offset that user can set for a specified GPU.
- Added nvmlDeviceGetTargetFanSpeed API to report the intended target speed of the device's specified fan.
- Added nvmlDeviceGetGpcClkMinMaxVfOffset API to report the Graphics clock min and max VF offset that user can set for a specified GPU.
- Added nvmlGpmMetricsGet to calculate GPM metrics from two GPM samples.
- Added nvmlGpmSampleFree to free allocated GPM sample.

- Added nvmlGpmSampleAlloc to allocate a GPM sample.
- Added nvmlGpmSampleGet to retrieve a GPM snapshot.
- Added nvmlGpmQueryDeviceSupport to query whether a device supports GPM
- Added **`nvmlDeviceGetSupportedPowerModes`** API to report the GPU's supported power mode mask.
- Added **`nvmlDeviceGetPowerMode`** API to report the GPU's current power mode.
- Added **`nvmlDeviceSetPowerMode`** API to set the new power mode.
- Added nvmlDeviceGetFanControlPolicy_v2 API to report the control policy for a specified GPU fan.
- Added nvmlDeviceSetFanControlPolicy API to set the control policy for a specified GPU fan.

Changes between v510 and v515

The following new functionality is exposed on NVIDIA display drivers version 515 Production or later.

- Added **`nvmlDeviceGetDefaultECCMode`** API to report the GPU's default ECC Mode.
- Added nvmlDeviceGetPcieSpeed API to report the GPU's PCIe link speed.
- Added nvmlDeviceGetDynamicPstatesInfo API to report the GPU's P-states information.
- Added nvmlDeviceSetFanSpeed_v2 API to set the GPU's fan speed.
- Added nvmlDeviceSetDefaultFanSpeed_v2 API to set the GPU's default fan speed.
- Added nvmlDeviceGetThermalSettings API to report the GPU's thermal system information.
- Added nvmlDeviceGetMinMaxClockOfPState API to report the min and max clocks of some clock domain for a given PState.
- Added nvmlDeviceGetSupportedPerformanceStates API to get all supported Performance States (P-States) for the GPU.
- Added nvmlDeviceGetGpcClkVfOffset API to report the GPCCLK VF offset value.
- Added nvmlDeviceSetGpcClkVfOffset API to set the GPCCLK VF offset value.
- Added nvmlDeviceGetMinMaxFanSpeed API to report the min and max fan speed that user can set for a specified GPU fan.

Changes between v495 and v510

The following new functionality is exposed on NVIDIA display drivers version 510 Production or later.

- Added nvmlDeviceGetGpuInstanceProfileInfoV and nvmlGpuInstanceGetComputeInstanceProfileInfoV APIs to include the profile name in their output.

- Added nvmlDeviceGetMemoryBusWidth API to report the GPU's Memory Bus Width.
- Added nvmlDeviceGetPcieLinkMaxSpeed API to report the GPU's PCIe Max Speed.
- Added nvmlDeviceGetPowerSource API to report the GPU's power source as AC or battery.
- Added **`nvmlDeviceGetNumberOfFans`** API to report the GPU's number of fans.
- Added nvmlDeviceGetNumGpuCores API to report the GPU's number of cores.
- Added nvmlDeviceGetMemoryInfo_v2. The new version accounts separately for system-reserved memory, and includes it in the used memory amount. The previous version of the API reduced the total memory amount by the amount of system-reserved memory.
- Added nvmlDeviceGetAdaptiveClockInfoStatus API to report the status of adaptive clocking for the GPU.

Changes between v465 and v470

The following new functionality is exposed on NVIDIA display drivers version 470 Production or later.

- Added new MIG GPU instance profile NVML_GPU_INSTANCE_PROFILE_1_SLICE_REV1.
- Added nvmlDeviceGetGpuInstancePossiblePlacements_v2. The previous version of the API will not support the profiles with possible placements greater than its total capacity, such as NVML_GPU_INSTANCE_PROFILE_1_SLICE_REV1.

Changes between v460 and v465

The following new functionality is exposed on NVIDIA display drivers version 465 Production or later.

- Added new NVML_BRAND_* enumeration values for NVIDIA, NVIDIA_RTX, GEFORCE_RTX, QUADRO_RTX and TITAN_RTX.
- Updated nvmlDeviceGetHandleByUUID to make it MIG-aware.
- Updated nvmlDeviceGetUUID to return MIG UUIDs in the canonical format, 'MIG-UUID'.
- Updated nvmlDeviceGetHandleByUUID to accept both UUID formats, 'MIG-UUID' and 'MIG-GPU UUID/GID/CID'.
- The nvmlDeviceSetAPIRestriction and nvmlDeviceGetAPIRestriction APIs would no longer support the ability to toggle root-only requirement for nvmlDeviceSetApplicationsClocks and nvmlDeviceResetApplicationsClocks.

Changes between v450 and v460

The following new functionality is exposed on NVIDIA display drivers version 460 Production or later.

- Added nvmlDeviceCreateGpuInstanceWithPlacement to allow placement specification when creating a new MIG GPU instance.

Changes between v445 and v450

The following new functionality is exposed on NVIDIA display drivers version 450 Production or later.

- Updated nvmlDeviceGetFanSpeed and nvmlDeviceGetFanSpeed_v2 for allowing fan speeds greater than 100% to be reported.
- Added nvmlDeviceGetCpuAffinityWithinScope to determine the closest processor(s) within a NUMA node or socket.
- Added nvmlDeviceGetMemoryAffinity to determine the closest NUMA node(s) within a NUMA node or socket.
- Added support to query and disable MIG mode on Windows.

Changes between v418 and v445

The following new functionality is exposed on NVIDIA display drivers version 445 Production or later.

- Added support for the NVIDIA Ampere architecture.
- Added support for Multi Instance GPU management. Refer to the "Multi Instance GPU Management" section for details.

Changes between v361 and v418

The following new functionality is exposed on NVIDIA display drivers version 418 Production or later.

- Added support for the Volta and Turing architectures, bug fixes, performance improvements, and new features.

Changes between v349 and v361

The following new functionality is exposed on NVIDIA display drivers version 361 Production or later.

- Added nvmlDeviceGetBoardPartNumber to return GPU part numbers
- Removed support for exclusive thread compute mode (Deprecated in 7.5)
- Added NVML_CLOCK_VIDEO (encoder/decoder) clock type as a supported clock type for nvmlDeviceGetClockInfo and nvmlDeviceGetMaxClockInfo.

Changes between v346 and v349

The following new functionality is exposed on NVIDIA display drivers version 349 Production or later.

- Added nvmlDeviceGetTopologyCommonAncestor to find the common path between two devices.
- Added nvmlDeviceGetTopologyNearestGpus to get a set of GPUs given a path level.
- Added nvmlSystemGetTopologyGpuSet to retrieve a set of GPUs with a given CPU affinity.
- Discontinued Perl bindings support.
- Updated nvmlDeviceGetAccountingPids , nvmlDeviceGetAccountingBufferSize and nvmlDeviceGetAccountingStats to report accounting information for both active and terminated processes. The execution time field in nvmlAccountingStats_t structure is populated only when the process is terminated.

Changes between v340 and v346

The following new functionality is exposed on NVIDIA display drivers version 346 Production or later.

- Added nvmlDeviceGetGraphicsRunningProcesses to get information about Graphics Processes running on a device.
- Added nvmlDeviceGetPcieReplayCounter to get PCI replay counters.
- Added nvmlDeviceGetPcieThroughput to get PCI utilization information.
- Discontinued Perl bindings support.

Changes between NVML v331 and v340

The following new functionality is exposed on NVIDIA display drivers version 340 Production or later.

- Added nvmlDeviceGetSamples to get recent power, utilization and clock samples for the GPU.
- Added nvmlDeviceGetTemperatureThreshold to get temperature thresholds for the GPU.
- Added nvmlDeviceGetBrand to get the brand name of the GPU.
- Added nvmlDeviceGetViolationStatus to get the duration of time during which the device was throttled (lower than requested clocks) due to power or thermal constraints. Violations due to thermal capping is not supported at this time.
- Added nvmlDeviceGetEncoderUtilization to get the GPU video encoder utilization.
- Added nvmlDeviceGetDecoderUtilization to get the GPU video decoder utilization.
- Added nvmlDeviceGetCpuAffinity to get the closest processor(s) affinity to a particular GPU.
- Added nvmlDeviceSetCpuAffinity to set the affinity of a particular GPU to the closest processor.
- Added nvmlDeviceClearCpuAffinity to clear the affinity of a particular GPU.
- Added nvmlDeviceGetBoardId to get a unique boardId for the running system.

- Added nvmlDeviceGetMultiGpuBoard to get whether the device is on a multiGPU board.
- Added nvmlDeviceGetAutoBoostedClocksEnabled and nvmlDeviceSetAutoBoostedClocksEnabled for querying and setting the state of auto boosted clocks on supporting hardware.
- Added nvmlDeviceSetDefaultAutoBoostedClocksEnabled for setting the default state of auto boosted clocks on supporting hardware.

Changes between NVML v5.319 Update and v331

The following new functionality is exposed on NVIDIA display drivers version 331 or later.

- Added nvmlDeviceGetMinorNumber to get the minor number for the device.
- Added nvmlDeviceGetBAR1MemoryInfo to get BAR1 total, available and used memory size.
- Added nvmlDeviceGetBridgeChipInfo to get the information related to bridge chip firmware.
- Added enforced power limit query API nvmlDeviceGetEnforcedPowerLimit
- Updated nvmlEventSetWait to return xid event data in case of xid error event.

Changes between NVML v5.319 RC and v5.319 Update

The following new functionality is exposed on NVIDIA display drivers version 319 Update or later.

- Added nvmlDeviceSetAPIRestriction and nvmlDeviceGetAPIRestriction, with initial ability to toggle root-only requirement for nvmlDeviceSetApplicationsClocks and nvmlDeviceResetApplicationsClocks.

Changes between NVML v4.304 Production and v5.319 RC

The following new functionality is exposed on NVIDIA display drivers version 319 RC or later.

- Added _v2 versions of nvmlDeviceGetHandleByIndex and nvmlDeviceGetCount that also count devices not accessible by current user
  - nvmlDeviceGetHandleByIndex_v2 (default) can also return NVML_ERROR_NO_PERMISSION
- Added nvmlInit_v2 and nvmlDeviceGetHandleByIndex_v2 that is safer and thus recommended function for initializing the library
  - nvmlInit_v2 lazily initializes only requested devices (queried with nvmlDeviceGetHandle*)
  - nvml.h defines nvmlInit_v2 and nvmlDeviceGetHandleByIndex_v2 as default functions

- Added nvmlDeviceGetIndex
- Added NVML_ERROR_GPU_IS_LOST to report GPUs that have fallen off the bus.
  - All NVML device APIs can return this error code, as a GPU can fall off the bus at any time.
- Added new class of APIs for gathering process statistics (nvmlAccountingStats)
- Application Clocks are no longer supported on GPU's from Quadro product line
- Added APIs to support dynamic page retirement. See nvmlDeviceGetRetiredPages and nvmlDeviceGetRetiredPagesPendingStatus
- Renamed nvmlClocksThrottleReasonUserDefinedClocks to nvmlClocksThrottleReasonApplicationsClocksSetting. Old name is deprecated and can be removed in one of the next major releases.
- Added nvmlDeviceGetDisplayActive and updated documentation to clarify how it differs from nvmlDeviceGetDisplayMode

Changes between NVML v4.304 RC and v4.304 Production

The following new functionality is exposed on NVIDIA display drivers version 304 Production or later.

- Added nvmlDeviceGetGpuOperationMode and nvmlDeviceSetGpuOperationMode.

Changes between NVML v3.295 and v4.304 RC

The following new functionality is exposed on NVIDIA display drivers version 304 RC or later.

- Added nvmlDeviceGetInforomConfigurationChecksum and nvmlDeviceValidateInforom.
- Added nvmlDeviceGetDisplayActive and updated documentation to clarify how it differs from nvmlDeviceGetDisplayMode.
- Added new error return value for initialization failure due to kernel module not receiving interrupts.
- Added nvmlDeviceSetApplicationsClocks, nvmlDeviceGetApplicationsClock, nvmlDeviceResetApplicationsClocks.
- Added nvmlDeviceGetSupportedMemoryClocks and nvmlDeviceGetSupportedGraphicsClocks.
- Added nvmlDeviceGetPowerManagementLimitConstraints, nvmlDeviceGetPowerManagementDefaultLimit and nvmlDeviceSetPowerManagementLimit.
- Added nvmlDeviceGetInforomImageVersion.
- Expanded nvmlDeviceGetUUID to support all CUDA capable GPUs.
- Deprecated nvmlDeviceGetDetailedEccErrors in favor of nvmlDeviceGetMemoryErrorCounter.

- Added NVML_MEMORY_LOCATION_TEXTURE_MEMORY to support reporting of texture memory error counters.
- Added nvmlDeviceGetCurrentClocksThrottleReasons and nvmlDeviceGetSupportedClocksThrottleReasons.
- NVML_CLOCK_SM is now also reported on supported Kepler devices.
- Dropped support for GT200 based Tesla brand GPUs: C1060, M1060, S1070.

Changes between NVML v2.285 and v3.295

The following new functionality is exposed on NVIDIA display drivers version 295 or later.

- Deprecated nvmlDeviceGetHandleBySerial in favor of newly added nvmlDeviceGetHandleByUUID.
- Marked the input parameters of nvmlDeviceGetHandleBySerial, nvmlDeviceGetHandleByUUID and nvmlDeviceGetHandleByPciBusId as const.
- Added nvmlDeviceOnSameBoard.
- Added nvmlConstants defines.
- Added nvmlDeviceGetMaxPcieLinkGeneration, nvmlDeviceGetMaxPcieLinkWidth, nvmlDeviceGetCurrPcieLinkGeneration,nvmlDeviceGetCurrPcieLinkWidth.
- Format change of nvmlDeviceGetUUID output to match the UUID standard. This function will return a different value.
- nvmlDeviceGetDetailedEccErrors will report zero for unsupported ECC error counters when a subset of ECC error counters are supported.

Changes between NVML v1.0 and v2.285

The following new functionality is exposed on NVIDIA display drivers version 285 or later.

- Added possibility to query separately current and pending driver model with nvmlDeviceGetDriverModel.
- Added API nvmlDeviceGetVbiosVersion function to report VBIOS version.
- Added pciSubSystemId to nvmlPciInfo_t struct.
- Added API nvmlErrorString function to convert error code to string.
- Updated docs to indicate we support M2075 and C2075.
- Added API nvmlSystemGetHicVersion function to report HIC firmware version.
- Added NVML versioning support
  - Functions that changed API and/or size of structs have appended versioning suffix (e.g., nvmlDeviceGetPciInfo_v2). Appropriate C defines have been added that map old function names to the newer version of the function.
- Added support for concurrent library usage by multiple libraries.

- Added API nvmlDeviceGetMaxClockInfo function for reporting device's clock limits.
- Added new error code NVML_ERROR_DRIVER_NOT_LOADED used by nvmlInit.
- Extended nvmlPciInfo_t struct with new field: sub system id.
- Added NVML support on Windows guest account.
- Changed format of pciBusId string (to XXXX:XX:XX.X) of nvmlPciInfo_t.
- Parsing of busId in nvmlDeviceGetHandleByPciBusId is less restrictive. You can pass 0:2:0.0 or 0000:02:00 and other variations.
- Added API for events waiting for GPU events (Linux only) see docs of nvmlEvents.
- Added API nvmlDeviceGetComputeRunningProcesses and nvmlSystemGetProcessName functions for looking up currently running compute applications.
- Deprecated nvmlDeviceGetPowerState in favor of nvmlDeviceGetPerformanceState.

Chapter 4. MODULES

Here is a list of all modules:

- vGPU Migration
- vGPU Utilization and Accounting
- Excluded GPU Queries
- Multi Instance GPU Management
- NVML GPM
  - GPM Enums
  - GPM Structs
  - GPM Functions
- VirtualGPU
  - vGPU Enums
  - vGPU Constants
  - vGPU Structs
- NvmlClocksEventReasons

4.1. Device Structs

struct nvmlPciInfoExt_v1_t

struct nvmlPciInfo_t

struct nvmlEccErrorCounts_t

struct nvmlUtilization_t

struct nvmlMemory_t

struct nvmlMemory_v2_t

struct nvmlBAR1Memory_t

struct nvmlProcessInfo_v1_t

struct nvmlProcessInfo_t

struct nvmlProcessDetail_v1_t

struct nvmlProcessDetailList_v1_t

struct nvmlC2cModeInfo_v1_t

struct nvmlRowRemapperHistogramValues_t

struct nvmlNvLinkUtilizationControl_t

struct nvmlBridgeChipInfo_t

struct nvmlBridgeChipHierarchy_t

union nvmlValue_t

struct nvmlSample_t

struct nvmlViolationTime_t

enum nvmlBridgeChipType_t

Enum to represent type of bridge chip

Values

NVML_BRIDGE_CHIP_PLX = 0
NVML_BRIDGE_CHIP_BRO4 = 1

enum nvmlNvLinkUtilizationCountUnits_t

Enum to represent the NvLink utilization counter packet units

Values

NVML_NVLINK_COUNTER_UNIT_CYCLES = 0
NVML_NVLINK_COUNTER_UNIT_PACKETS = 1
NVML_NVLINK_COUNTER_UNIT_BYTES = 2
NVML_NVLINK_COUNTER_UNIT_RESERVED = 3
NVML_NVLINK_COUNTER_UNIT_COUNT

enum nvmlNvLinkUtilizationCountPktTypes_t

Enum to represent the NvLink utilization counter packet types to count ** this is ONLY applicable with the units as packets or bytes ** as specified in nvmlNvLinkUtilizationCountUnits_t ** all packet filter descriptions are target GPU centric ** these can be "OR'd" together

Values

NVML_NVLINK_COUNTER_PKTFILTER_NOP = 0x1
NVML_NVLINK_COUNTER_PKTFILTER_READ = 0x2
NVML_NVLINK_COUNTER_PKTFILTER_WRITE = 0x4
NVML_NVLINK_COUNTER_PKTFILTER_RATOM = 0x8
NVML_NVLINK_COUNTER_PKTFILTER_NRATOM = 0x10
NVML_NVLINK_COUNTER_PKTFILTER_FLUSH = 0x20
NVML_NVLINK_COUNTER_PKTFILTER_RESPDATA = 0x40
NVML_NVLINK_COUNTER_PKTFILTER_RESPNODATA = 0x80
NVML_NVLINK_COUNTER_PKTFILTER_ALL = 0xFF

enum nvmlNvLinkCapability_t

Enum to represent NvLink queryable capabilities

Values

NVML_NVLINK_CAP_P2P_SUPPORTED = 0
NVML_NVLINK_CAP_SYSMEM_ACCESS = 1
NVML_NVLINK_CAP_P2P_ATOMICS = 2
NVML_NVLINK_CAP_SYSMEM_ATOMICS = 3
NVML_NVLINK_CAP_SLI_BRIDGE = 4
NVML_NVLINK_CAP_VALID = 5
NVML_NVLINK_CAP_COUNT

enum nvmlNvLinkErrorCounter_t

Enum to represent NvLink queryable error counters

Values

NVML_NVLINK_ERROR_DL_REPLAY = 0
NVML_NVLINK_ERROR_DL_RECOVERY = 1
NVML_NVLINK_ERROR_DL_CRC_FLIT = 2
NVML_NVLINK_ERROR_DL_CRC_DATA = 3
NVML_NVLINK_ERROR_DL_ECC_DATA = 4
NVML_NVLINK_ERROR_COUNT

enum nvmlIntNvLinkDeviceType_t

Enum to represent NvLink's remote device type

Values

NVML_NVLINK_DEVICE_TYPE_GPU = 0x00
NVML_NVLINK_DEVICE_TYPE_IBMNPU = 0x01
NVML_NVLINK_DEVICE_TYPE_SWITCH = 0x02
NVML_NVLINK_DEVICE_TYPE_UNKNOWN = 0xFF

enum nvmlGpuTopologyLevel_t

Represents level relationships within a system between two GPUs The enums are spaced to allow for future relationships

Values

NVML_TOPOLOGY_INTERNAL = 0
NVML_TOPOLOGY_SINGLE = 10
NVML_TOPOLOGY_MULTIPLE = 20
NVML_TOPOLOGY_HOSTBRIDGE = 30
NVML_TOPOLOGY_NODE = 40
NVML_TOPOLOGY_SYSTEM = 50

enum nvmlSamplingType_t

Represents Type of Sampling Event

Values

NVML_TOTAL_POWER_SAMPLES = 0
To represent total power drawn by GPU.
NVML_GPU_UTILIZATION_SAMPLES = 1
To represent percent of time during which one or more kernels was executing on the GPU.
NVML_MEMORY_UTILIZATION_SAMPLES = 2
To represent percent of time during which global (device) memory was being read or written.
NVML_ENC_UTILIZATION_SAMPLES = 3
To represent percent of time during which NVENC remains busy.
NVML_DEC_UTILIZATION_SAMPLES = 4
To represent percent of time during which NVDEC remains busy.
NVML_PROCESSOR_CLK_SAMPLES = 5
To represent processor clock samples.
NVML_MEMORY_CLK_SAMPLES = 6
To represent memory clock samples.
NVML_MODULE_POWER_SAMPLES = 7
To represent module power samples for total module starting Grace Hopper.
NVML_JPG_UTILIZATION_SAMPLES = 8
To represent percent of time during which NVJPG remains busy.
NVML_OFA_UTILIZATION_SAMPLES = 9
To represent percent of time during which NVOFA remains busy.
NVML_SAMPLINGTYPE_COUNT

enum nvmlPcieUtilCounter_t

Represents the queryable PCIe utilization counters

Values

NVML_PCIE_UTIL_TX_BYTES = 0
NVML_PCIE_UTIL_RX_BYTES = 1
NVML_PCIE_UTIL_COUNT

enum nvmlValueType_t

Represents the type for sample value returned

Values

NVML_VALUE_TYPE_DOUBLE = 0
NVML_VALUE_TYPE_UNSIGNED_INT = 1
NVML_VALUE_TYPE_UNSIGNED_LONG = 2
NVML_VALUE_TYPE_UNSIGNED_LONG_LONG = 3
NVML_VALUE_TYPE_SIGNED_LONG_LONG = 4
NVML_VALUE_TYPE_SIGNED_INT = 5
NVML_VALUE_TYPE_COUNT

enum nvmlPerfPolicyType_t

Represents type of perf policy for which violation times can be queried

Values

NVML_PERF_POLICY_POWER = 0
How long did power violations cause the GPU to be below application clocks.
NVML_PERF_POLICY_THERMAL = 1
How long did thermal violations cause the GPU to be below application clocks.
NVML_PERF_POLICY_SYNC_BOOST = 2
How long did sync boost cause the GPU to be below application clocks.
NVML_PERF_POLICY_BOARD_LIMIT = 3
How long did the board limit cause the GPU to be below application clocks.
NVML_PERF_POLICY_LOW_UTILIZATION = 4
How long did low utilization cause the GPU to be below application clocks.
NVML_PERF_POLICY_RELIABILITY = 5
How long did the board reliability limit cause the GPU to be below application clocks.
NVML_PERF_POLICY_TOTAL_APP_CLOCKS = 10
Total time the GPU was held below application clocks by any limiter (0 - 5 above).
NVML_PERF_POLICY_TOTAL_BASE_CLOCKS = 11
Total time the GPU was held below base clocks.
NVML_PERF_POLICY_COUNT

#define NVML_VALUE_NOT_AVAILABLE (-1)

Special constant that some fields take when they are not available. Used when only part of the struct is not available.

Each structure explicitly states when to check for this value.

#define NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE 32

Buffer size guaranteed to be large enough for pci bus id

#define NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE 16

Buffer size guaranteed to be large enough for pci bus id for busIdLegacy

#define NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT "%04X:%02X:%02X.0"

PCI format string for busIdLegacy

#define NVML_DEVICE_PCI_BUS_ID_FMT "%08X:%02X:%02X.0"

PCI format string for busId

#define NVML_DEVICE_PCI_BUS_ID_FMT_ARGS (pciInfo)->domain, \ (pciInfo)->bus, \ (pciInfo)->device

Utility macro for filling the pci bus id format from a nvmlPciInfo_t

#define nvmlProcessDetailList_v1 NVML_STRUCT_VERSION(ProcessDetailList, 1)

nvmlProcessDetailList version

#define NVML_NVLINK_MAX_LINKS 18

Maximum number of NvLink links supported

#define NVML_MAX_PHYSICAL_BRIDGE (128)

Maximum limit on Physical Bridges per Board

4.2. Device Enums

struct nvmlClkMonFaultInfo_t

struct nvmlClkMonStatus_t

enum nvmlEnableState_t

Generic enable/disable enum.

Values

NVML_FEATURE_DISABLED = 0
Feature disabled.
NVML_FEATURE_ENABLED = 1
Feature enabled.

enum nvmlBrandType_t

* The Brand of the GPU

Values

NVML_BRAND_UNKNOWN = 0
NVML_BRAND_QUADRO = 1
NVML_BRAND_TESLA = 2
NVML_BRAND_NVS = 3
NVML_BRAND_GRID = 4
NVML_BRAND_GEFORCE = 5
NVML_BRAND_TITAN = 6
NVML_BRAND_NVIDIA_VAPPS = 7
NVML_BRAND_NVIDIA_VPC = 8
NVML_BRAND_NVIDIA_VCS = 9
NVML_BRAND_NVIDIA_VWS = 10
NVML_BRAND_NVIDIA_CLOUD_GAMING = 11
NVML_BRAND_NVIDIA_VGAMING =
NVML_BRAND_NVIDIA_CLOUD_GAMING
NVML_BRAND_QUADRO_RTX = 12
NVML_BRAND_NVIDIA_RTX = 13
NVML_BRAND_NVIDIA = 14
NVML_BRAND_GEFORCE_RTX = 15
NVML_BRAND_TITAN_RTX = 16
NVML_BRAND_COUNT

enum nvmlTemperatureThresholds_t

Temperature thresholds.

Values

NVML_TEMPERATURE_THRESHOLD_SHUTDOWN = 0
NVML_TEMPERATURE_THRESHOLD_SLOWDOWN = 1
NVML_TEMPERATURE_THRESHOLD_MEM_MAX = 2
NVML_TEMPERATURE_THRESHOLD_GPU_MAX = 3
NVML_TEMPERATURE_THRESHOLD_ACOUSTIC_MIN = 4
NVML_TEMPERATURE_THRESHOLD_ACOUSTIC_CURR = 5
NVML_TEMPERATURE_THRESHOLD_ACOUSTIC_MAX = 6
NVML_TEMPERATURE_THRESHOLD_COUNT

enum nvmlTemperatureSensors_t

Temperature sensors.

Values

NVML_TEMPERATURE_GPU = 0
Temperature sensor for the GPU die.
NVML_TEMPERATURE_COUNT

enum nvmlComputeMode_t

Compute mode.

NVML_COMPUTEMODE_EXCLUSIVE_PROCESS was added in CUDA 4.0. Earlier CUDA versions supported a single exclusive mode, which is equivalent to NVML_COMPUTEMODE_EXCLUSIVE_THREAD in CUDA 4.0 and beyond.

Values

NVML_COMPUTEMODE_DEFAULT = 0
Default compute mode -- multiple contexts per device.
NVML_COMPUTEMODE_EXCLUSIVE_THREAD = 1
Support Removed.
NVML_COMPUTEMODE_PROHIBITED = 2
Compute-prohibited mode -- no contexts per device.
NVML_COMPUTEMODE_EXCLUSIVE_PROCESS = 3
Compute-exclusive-process mode -- only one context per device, usable from multiple threads at a time.
NVML_COMPUTEMODE_COUNT

enum nvmlMemoryErrorType_t

Memory error types

Values

NVML_MEMORY_ERROR_TYPE_CORRECTED = 0
A memory error that was correctedFor ECC errors, these are single bit errors For Texture memory, these are errors fixed by resend

NVML_MEMORY_ERROR_TYPE_UNCORRECTED = 1
A memory error that was not correctedFor ECC errors, these are double bit errors For Texture memory, these are errors where the resend fails

NVML_MEMORY_ERROR_TYPE_COUNT
Count of memory error types.

enum nvmlEccCounterType_t

ECC counter types.

Note: Volatile counts are reset each time the driver loads. On Windows this is once per boot. On Linux this can be more frequent. On Linux the driver unloads when no active clients exist. If persistence mode is enabled or there is always a driver client active (e.g. X11), then Linux also sees per-boot behavior. If not, volatile counts are reset each time a compute app is run.

Values

NVML_VOLATILE_ECC = 0
Volatile counts are reset each time the driver loads.

NVML_AGGREGATE_ECC = 1
Aggregate counts persist across reboots (i.e. for the lifetime of the device).

NVML_ECC_COUNTER_TYPE_COUNT
Count of memory counter types.

enum nvmlClockType_t

Clock types.

All speeds are in Mhz.

Values

NVML_CLOCK_GRAPHICS = 0
Graphics clock domain.

NVML_CLOCK_SM = 1
SM clock domain.

NVML_CLOCK_MEM = 2
Memory clock domain.
NVML_CLOCK_VIDEO = 3
Video encoder/decoder clock domain.
NVML_CLOCK_COUNT
Count of clock types.

enum nvmlClockId_t

Clock Ids. These are used in combination with nvmlClockType_t to specify a single clock value.

Values

NVML_CLOCK_ID_CURRENT = 0
Current actual clock value.
NVML_CLOCK_ID_APP_CLOCK_TARGET = 1
Target application clock.
NVML_CLOCK_ID_APP_CLOCK_DEFAULT = 2
Default application clock target.
NVML_CLOCK_ID_CUSTOMER_BOOST_MAX = 3
OEM-defined maximum clock rate.
NVML_CLOCK_ID_COUNT
Count of Clock Ids.

enum nvmlDriverModel_t

Driver models.

Windows only.

Values

NVML_DRIVER_WDDM = 0
WDDM driver model -- GPU treated as a display device.
NVML_DRIVER_WDM = 1
WDM (TCC) model (recommended) -- GPU treated as a generic device.

enum nvmlPstates_t

Allowed PStates.

Values

NVML_PSTATE_0 = 0
Performance state 0 -- Maximum Performance.

NVML_PSTATE_1 = 1
Performance state 1.
NVML_PSTATE_2 = 2
Performance state 2.
NVML_PSTATE_3 = 3
Performance state 3.
NVML_PSTATE_4 = 4
Performance state 4.
NVML_PSTATE_5 = 5
Performance state 5.
NVML_PSTATE_6 = 6
Performance state 6.
NVML_PSTATE_7 = 7
Performance state 7.
NVML_PSTATE_8 = 8
Performance state 8.
NVML_PSTATE_9 = 9
Performance state 9.
NVML_PSTATE_10 = 10
Performance state 10.
NVML_PSTATE_11 = 11
Performance state 11.
NVML_PSTATE_12 = 12
Performance state 12.
NVML_PSTATE_13 = 13
Performance state 13.
NVML_PSTATE_14 = 14
Performance state 14.
NVML_PSTATE_15 = 15
Performance state 15 -- Minimum Performance.
NVML_PSTATE_UNKNOWN = 32
Unknown performance state.

enum nvmlGpuOperationMode_t

GPU Operation Mode

GOM allows to reduce power usage and optimize GPU throughput by disabling GPU features.

Each GOM is designed to meet specific user needs.

Values

NVML_GOM_ALL_ON = 0
Everything is enabled and running at full speed.
NVML_GOM_COMPUTE = 1
Designed for running only compute tasks. Graphics operations are not allowed
NVML_GOM_LOW_DP = 2
Designed for running graphics applications that don't require high bandwidth double precision

enum nvmlInforomObject_t

Available infoROM objects.

Values

NVML_INFOROM_OEM = 0
An object defined by OEM.
NVML_INFOROM_ECC = 1
The ECC object determining the level of ECC support.
NVML_INFOROM_POWER = 2
The power management object.
NVML_INFOROM_COUNT
This counts the number of infoROM objects the driver knows about.

enum nvmlReturn_t

Return values for NVML API calls.

Values

NVML_SUCCESS = 0
The operation was successful.
NVML_ERROR_UNINITIALIZED = 1
NVML was not first initialized with nvmlInit().
NVML_ERROR_INVALID_ARGUMENT = 2
A supplied argument is invalid.
NVML_ERROR_NOT_SUPPORTED = 3
The requested operation is not available on target device.
NVML_ERROR_NO_PERMISSION = 4
The current user does not have permission for operation.
NVML_ERROR_ALREADY_INITIALIZED = 5
Deprecated: Multiple initializations are now allowed through ref counting.
NVML_ERROR_NOT_FOUND = 6
A query to find an object was unsuccessful.

NVML_ERROR_INSUFFICIENT_SIZE = 7
An input argument is not large enough.
NVML_ERROR_INSUFFICIENT_POWER = 8
A device's external power cables are not properly attached.
NVML_ERROR_DRIVER_NOT_LOADED = 9
NVIDIA driver is not loaded.
NVML_ERROR_TIMEOUT = 10
User provided timeout passed.
NVML_ERROR_IRQ_ISSUE = 11
NVIDIA Kernel detected an interrupt issue with a GPU.
NVML_ERROR_LIBRARY_NOT_FOUND = 12
NVML Shared Library couldn't be found or loaded.
NVML_ERROR_FUNCTION_NOT_FOUND = 13
Local version of NVML doesn't implement this function.
NVML_ERROR_CORRUPTED_INFOROM = 14
infoROM is corrupted
NVML_ERROR_GPU_IS_LOST = 15
The GPU has fallen off the bus or has otherwise become inaccessible.
NVML_ERROR_RESET_REQUIRED = 16
The GPU requires a reset before it can be used again.
NVML_ERROR_OPERATING_SYSTEM = 17
The GPU control device has been blocked by the operating system/cgroups.
NVML_ERROR_LIB_RM_VERSION_MISMATCH = 18
RM detects a driver/library version mismatch.
NVML_ERROR_IN_USE = 19
An operation cannot be performed because the GPU is currently in use.
NVML_ERROR_MEMORY = 20
Insufficient memory.
NVML_ERROR_NO_DATA = 21
No data.
NVML_ERROR_VGPU_ECC_NOT_SUPPORTED = 22
The requested vgpu operation is not available on target device, becasue ECC is enabled.
NVML_ERROR_INSUFFICIENT_RESOURCES = 23
Ran out of critical resources, other than memory.
NVML_ERROR_FREQ_NOT_SUPPORTED = 24
Ran out of critical resources, other than memory.
NVML_ERROR_ARGUMENT_VERSION_MISMATCH = 25
The provided version is invalid/unsupported.
NVML_ERROR_DEPRECATED = 26
The requested functionality has been deprecated.
NVML_ERROR_NOT_READY = 27
The system is not ready for the request.

NVML_ERROR_GPU_NOT_FOUND = 28
No GPUs were found.
NVML_ERROR_INVALID_STATE = 29
Resource not in correct state to perform requested operation.
NVML_ERROR_UNKNOWN = 999
An internal driver error occurred.

enum nvmlMemoryLocation_t

See nvmlDeviceGetMemoryErrorCounter

Values

NVML_MEMORY_LOCATION_L1_CACHE = 0
GPU L1 Cache.
NVML_MEMORY_LOCATION_L2_CACHE = 1
GPU L2 Cache.
NVML_MEMORY_LOCATION_DRAM = 2
Turing+ DRAM.
NVML_MEMORY_LOCATION_DEVICE_MEMORY = 2
GPU Device Memory.
NVML_MEMORY_LOCATION_REGISTER_FILE = 3
GPU Register File.
NVML_MEMORY_LOCATION_TEXTURE_MEMORY = 4
GPU Texture Memory.
NVML_MEMORY_LOCATION_TEXTURE_SHM = 5
Shared memory.
NVML_MEMORY_LOCATION_CBU = 6
CBU.
NVML_MEMORY_LOCATION_SRAM = 7
Turing+ SRAM.
NVML_MEMORY_LOCATION_COUNT
This counts the number of memory locations the driver knows about.

enum nvmlPageRetirementCause_t

Causes for page retirement

Values

NVML_PAGE_RETIREMENT_CAUSE_MULTIPLE_SINGLE_BIT_ECC_ERRORS = 0
Page was retired due to multiple single bit ECC error.
NVML_PAGE_RETIREMENT_CAUSE_DOUBLE_BIT_ECC_ERROR = 1
Page was retired due to double bit ECC error.
NVML_PAGE_RETIREMENT_CAUSE_COUNT

enum nvmlRestrictedAPI_t

API types that allow changes to default permission restrictions

Values

NVML_RESTRICTED_API_SET_APPLICATION_CLOCKS = 0
APIs that change application clocks, see nvmlDeviceSetApplicationsClocks and see nvmlDeviceResetApplicationsClocks

NVML_RESTRICTED_API_SET_AUTO_BOOSTED_CLOCKS = 1
APIs that enable/disable Auto Boosted clocks see nvmlDeviceSetAutoBoostedClocksEnabled

NVML_RESTRICTED_API_COUNT

#define nvmlFlagDefault 0x00

Generic flag used to specify the default behavior of some functions. See description of particular functions for details.

#define nvmlFlagForce 0x01

Generic flag used to force some behavior. See description of particular functions for details.

#define MAX_CLK_DOMAINS 32

Max Clock Monitors available

#define nvmlEccBitType_t nvmlMemoryErrorType_t

ECC bit types.

Deprecated See nvmlMemoryErrorType_t for a more flexible type

#define NVML_SINGLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_CORRECTED

Single bit ECC errors

Deprecated Mapped to NVML_MEMORY_ERROR_TYPE_CORRECTED

#define NVML_DOUBLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_UNCORRECTED

Double bit ECC errors

Deprecated Mapped to NVML_MEMORY_ERROR_TYPE_UNCORRECTED

4.3. Field Value Enums

struct nvmlFieldValue_t

#define NVML_FI_DEV_ECC_CURRENT 1

Current ECC mode. 1=Active. 0=Inactive.

Field Identifiers.

All Identifiers pertain to a device. Each ID is only used once and is guaranteed never to change.

#define NVML_FI_DEV_ECC_PENDING 2

Pending ECC mode. 1=Active. 0=Inactive.

#define NVML_FI_DEV_ECC_SBE_VOL_TOTAL 3

Total single bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_TOTAL 4

Total double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_SBE_AGG_TOTAL 5

Total single bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_TOTAL 6

Total double bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_SBE_VOL_L1 7

L1 cache single bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_L1 8

L1 cache double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_SBE_VOL_L2 9

L2 cache single bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_L2 10

L2 cache double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_SBE_VOL_DEV 11

Device memory single bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_DEV 12

Device memory double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_SBE_VOL_REG 13

Register file single bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_REG 14

Register file double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_SBE_VOL_TEX 15

Texture memory single bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_TEX 16

Texture memory double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_CBU 17

CBU double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_SBE_AGG_L1 18

L1 cache single bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_L1 19

L1 cache double bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_SBE_AGG_L2 20

L2 cache single bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_L2 21

L2 cache double bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_SBE_AGG_DEV 22

Device memory single bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_DEV 23

Device memory double bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_SBE_AGG_REG 24

Register File single bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_REG 25

Register File double bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_SBE_AGG_TEX 26

Texture memory single bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_TEX 27

Texture memory double bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_CBU 28

CBU double bit aggregate ECC errors.

#define NVML_FI_DEV_RETIRED_SBE 29

Number of retired pages because of single bit errors.

#define NVML_FI_DEV_RETIRED_DBE 30

Number of retired pages because of double bit errors.

#define NVML_FI_DEV_RETIRED_PENDING 31

If any pages are pending retirement. 1=yes. 0=no.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L0 32

NVLink flow control CRC Error Counter for Lane 0.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L1 33

NVLink flow control CRC Error Counter for Lane 1.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L2 34

NVLink flow control CRC Error Counter for Lane 2.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L3 35

NVLink flow control CRC Error Counter for Lane 3.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L4 36

NVLink flow control CRC Error Counter for Lane 4.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L5 37

NVLink flow control CRC Error Counter for Lane 5.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_TOTAL 38

NVLink flow control CRC Error Counter total for all Lanes.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L0 39

NVLink data CRC Error Counter for Lane 0.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L1 40

NVLink data CRC Error Counter for Lane 1.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L2 41

NVLink data CRC Error Counter for Lane 2.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L3 42

NVLink data CRC Error Counter for Lane 3.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L4 43

NVLink data CRC Error Counter for Lane 4.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L5 44

NVLink data CRC Error Counter for Lane 5.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_TOTAL 45

NvLink data CRC Error Counter total for all Lanes.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L0 46

NVLink Replay Error Counter for Lane 0.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L1 47

NVLink Replay Error Counter for Lane 1.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L2 48

NVLink Replay Error Counter for Lane 2.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L3 49

NVLink Replay Error Counter for Lane 3.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L4 50

NVLink Replay Error Counter for Lane 4.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L5 51

NVLink Replay Error Counter for Lane 5.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_TOTAL 52

NVLink Replay Error Counter total for all Lanes.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L0 53

NVLink Recovery Error Counter for Lane 0.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L1 54

NVLink Recovery Error Counter for Lane 1.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L2 55

NVLink Recovery Error Counter for Lane 2.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L3 56

NVLink Recovery Error Counter for Lane 3.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L4 57

NVLink Recovery Error Counter for Lane 4.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L5 58

NVLink Recovery Error Counter for Lane 5.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_TOTAL 59

NVLink Recovery Error Counter total for all Lanes.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L0 60

NVLink Bandwidth Counter for Counter Set 0, Lane 0.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L1 61

NVLink Bandwidth Counter for Counter Set 0, Lane 1.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L2 62

NVLink Bandwidth Counter for Counter Set 0, Lane 2.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L3 63

NVLink Bandwidth Counter for Counter Set 0, Lane 3.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L4 64

NVLink Bandwidth Counter for Counter Set 0, Lane 4.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L5 65

NVLink Bandwidth Counter for Counter Set 0, Lane 5.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_TOTAL 66

NVLink Bandwidth Counter Total for Counter Set 0, All Lanes.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L0 67

NVLink Bandwidth Counter for Counter Set 1, Lane 0.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L1 68

NVLink Bandwidth Counter for Counter Set 1, Lane 1.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L2 69

NVLink Bandwidth Counter for Counter Set 1, Lane 2.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L3 70

NVLink Bandwidth Counter for Counter Set 1, Lane 3.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L4 71

NVLink Bandwidth Counter for Counter Set 1, Lane 4.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L5 72

NVLink Bandwidth Counter for Counter Set 1, Lane 5.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_TOTAL 73

NVLink Bandwidth Counter Total for Counter Set 1, All Lanes.

#define NVML_FI_DEV_PERF_POLICY_POWER 74

Perf Policy Counter for Power Policy.

#define NVML_FI_DEV_PERF_POLICY_THERMAL 75

Perf Policy Counter for Thermal Policy.

#define NVML_FI_DEV_PERF_POLICY_SYNC_BOOST 76

Perf Policy Counter for Sync boost Policy.

#define NVML_FI_DEV_PERF_POLICY_BOARD_LIMIT 77

Perf Policy Counter for Board Limit.

#define NVML_FI_DEV_PERF_POLICY_LOW_UTILIZATION 78

Perf Policy Counter for Low GPU Utilization Policy.

#define NVML_FI_DEV_PERF_POLICY_RELIABILITY 79

Perf Policy Counter for Reliability Policy.

#define NVML_FI_DEV_PERF_POLICY_TOTAL_APP_CLOCKS 80

Perf Policy Counter for Total App Clock Policy.

#define NVML_FI_DEV_PERF_POLICY_TOTAL_BASE_CLOCKS 81

Perf Policy Counter for Total Base Clocks Policy.

#define NVML_FI_DEV_MEMORY_TEMP 82

Memory temperature for the device.

#define NVML_FI_DEV_TOTAL_ENERGY_CONSUMPTION 83

Total energy consumption for the GPU in mJ since the driver was last reloaded.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L0 84

NVLink Speed in MBps for Link 0.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L1 85

NVLink Speed in MBps for Link 1.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L2 86

NVLink Speed in MBps for Link 2.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L3 87

NVLink Speed in MBps for Link 3.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L4 88

NVLink Speed in MBps for Link 4.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L5 89

NVLink Speed in MBps for Link 5.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_COMMON 90

Common NVLink Speed in MBps for active links.

#define NVML_FI_DEV_NVLINK_LINK_COUNT 91

Number of NVLinks present on the device.

#define NVML_FI_DEV_RETIRED_PENDING_SBE 92

If any pages are pending retirement due to SBE. 1=yes. 0=no.

#define NVML_FI_DEV_RETIRED_PENDING_DBE 93

If any pages are pending retirement due to DBE. 1=yes. 0=no.

#define NVML_FI_DEV_PCIE_REPLAY_COUNTER 94

PCIe replay counter.

#define NVML_FI_DEV_PCIE_REPLAY_ROLLOVER_COUNTER 95

PCIe replay rollover counter.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L6 96

NVLink flow control CRC Error Counter for Lane 6.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L7 97

NVLink flow control CRC Error Counter for Lane 7.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L8 98

NVLink flow control CRC Error Counter for Lane 8.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L9 99

NVLink flow control CRC Error Counter for Lane 9.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L10 100

NVLink flow control CRC Error Counter for Lane 10.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L11 101

NVLink flow control CRC Error Counter for Lane 11.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L6 102

NVLink data CRC Error Counter for Lane 6.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L7 103

NVLink data CRC Error Counter for Lane 7.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L8 104

NVLink data CRC Error Counter for Lane 8.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L9 105

NVLink data CRC Error Counter for Lane 9.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L10 106

NVLink data CRC Error Counter for Lane 10.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L11 107

NVLink data CRC Error Counter for Lane 11.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L6 108

NVLink Replay Error Counter for Lane 6.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L7 109

NVLink Replay Error Counter for Lane 7.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L8 110

NVLink Replay Error Counter for Lane 8.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L9 111

NVLink Replay Error Counter for Lane 9.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L10 112

NVLink Replay Error Counter for Lane 10.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L11 113

NVLink Replay Error Counter for Lane 11.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L6 114

NVLink Recovery Error Counter for Lane 6.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L7 115

NVLink Recovery Error Counter for Lane 7.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L8 116

NVLink Recovery Error Counter for Lane 8.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L9 117

NVLink Recovery Error Counter for Lane 9.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L10 118

NVLink Recovery Error Counter for Lane 10.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L11 119

NVLink Recovery Error Counter for Lane 11.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L6 120

NVLink Bandwidth Counter for Counter Set 0, Lane 6.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L7 121

NVLink Bandwidth Counter for Counter Set 0, Lane 7.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L8 122

NVLink Bandwidth Counter for Counter Set 0, Lane 8.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L9 123

NVLink Bandwidth Counter for Counter Set 0, Lane 9.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L10 124

NVLink Bandwidth Counter for Counter Set 0, Lane 10.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L11 125

NVLink Bandwidth Counter for Counter Set 0, Lane 11.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L6 126

NVLink Bandwidth Counter for Counter Set 1, Lane 6.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L7 127

NVLink Bandwidth Counter for Counter Set 1, Lane 7.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L8 128

NVLink Bandwidth Counter for Counter Set 1, Lane 8.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L9 129

NVLink Bandwidth Counter for Counter Set 1, Lane 9.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L10 130

NVLink Bandwidth Counter for Counter Set 1, Lane 10.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L11 131

NVLink Bandwidth Counter for Counter Set 1, Lane 11.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L6 132

NVLink Speed in MBps for Link 6.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L7 133

NVLink Speed in MBps for Link 7.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L8 134

NVLink Speed in MBps for Link 8.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L9 135

NVLink Speed in MBps for Link 9.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L10 136

NVLink Speed in MBps for Link 10.

#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L11 137

NVLink Speed in MBps for Link 11.

#define NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_TX 138

NVLink TX Data throughput in KiB.

NVLink throughput counters field values

Link ID needs to be specified in the scopeId field in nvmlFieldValue_t. A scopeId of UINT_MAX returns aggregate value summed up across all links for the specified counter type in fieldId.

#define NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_RX 139

NVLink RX Data throughput in KiB.

#define NVML_FI_DEV_NVLINK_THROUGHPUT_RAW_TX 140

NVLink TX Data + protocol overhead in KiB.

#define NVML_FI_DEV_NVLINK_THROUGHPUT_RAW_RX 141

NVLink RX Data + protocol overhead in KiB.

#define NVML_FI_DEV_REMAPPED_COR 142

Number of remapped rows due to correctable errors.

#define NVML_FI_DEV_REMAPPED_UNC 143

Number of remapped rows due to uncorrectable errors.

#define NVML_FI_DEV_REMAPPED_PENDING 144

If any rows are pending remapping. 1=yes 0=no.

#define NVML_FI_DEV_REMAPPED_FAILURE 145

If any rows failed to be remapped 1=yes 0=no.

#define NVML_FI_DEV_NVLINK_REMOTE_NVLINK_ID 146

Remote device NVLink ID.

Remote device NVLink ID

Link ID needs to be specified in the scopeId field in nvmlFieldValue_t.

#define NVML_FI_DEV_NVSWITCH_CONNECTED_LINK_COUNT 147

Number of NVLinks connected to NVSwitch.

NVSwitch: connected NVLink count

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L0 148

NVLink data ECC Error Counter for Link 0.

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L1 149

NVLink data ECC Error Counter for Link 1.

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L2 150

NVLink data ECC Error Counter for Link 2.

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L3 151

NVLink data ECC Error Counter for Link 3.

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L4 152

NVLink data ECC Error Counter for Link 4.

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L5 153

NVLink data ECC Error Counter for Link 5.

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L6 154

NVLink data ECC Error Counter for Link 6.

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L7 155

NVLink data ECC Error Counter for Link 7.

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L8 156

NVLink data ECC Error Counter for Link 8.

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L9 157

NVLink data ECC Error Counter for Link 9.

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L10 158

NVLink data ECC Error Counter for Link 10.

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L11 159

NVLink data ECC Error Counter for Link 11.

#define NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_TOTAL 160

NVLink data ECC Error Counter total for all Links.

#define NVML_FI_DEV_NVLINK_ERROR_DL_REPLAY 161

NVLink Replay Error Counter.

#define NVML_FI_DEV_NVLINK_ERROR_DL_RECOVERY 162

NVLink Recovery Error Counter.

#define NVML_FI_DEV_NVLINK_ERROR_DL_CRC 163

NVLink CRC Error Counter.

#define NVML_FI_DEV_NVLINK_GET_SPEED 164

NVLink Speed in MBps.

#define NVML_FI_DEV_NVLINK_GET_STATE 165

NVLink State - Active,Inactive.

#define NVML_FI_DEV_NVLINK_GET_VERSION 166

NVLink Version.

#define NVML_FI_DEV_NVLINK_GET_POWER_STATE 167

NVLink Power state. 0=HIGH_SPEED 1=LOW_SPEED.

#define NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD 168

NVLink length of idle period (in units of 100us) before transitioning links to sleep state.

#define NVML_FI_DEV_PCIE_L0_TO_RECOVERY_COUNTER 169

Device PEX error recovery counter.

#define NVML_FI_DEV_C2C_LINK_COUNT 170

Number of C2C Links present on the device.

#define NVML_FI_DEV_C2C_LINK_GET_STATUS 171

C2C Link Status 0=INACTIVE 1=ACTIVE.

#define NVML_FI_DEV_C2C_LINK_GET_MAX_BW 172

C2C Link Speed in MBps for active links.

#define NVML_FI_DEV_POWER_AVERAGE 185

GPU power averaged over 1 sec interval, supported on Ampere (except GA100) or newer architectures.

Retrieves power usage for this GPU in milliwatts. It is only available if power management mode is supported. See nvmlDeviceGetPowerManagementMode and nvmlDeviceGetPowerUsage.

scopeId needs to be specified. It signifies: 0 - GPU Only Scope - Metrics for GPU are retrieved 1 - Module scope - Metrics for the module (e.g. CPU + GPU) are retrieved. Note: CPU here refers to NVIDIA CPU (e.g. Grace). x86 or non-NVIDIA ARM is not supported

#define NVML_FI_DEV_POWER_INSTANT 186

Current GPU power, supported on all architectures.

#define NVML_FI_DEV_POWER_MIN_LIMIT 187

Minimum power limit in milliwatts.

#define NVML_FI_DEV_POWER_MAX_LIMIT 188

Maximum power limit in milliwatts.

#define NVML_FI_DEV_POWER_DEFAULT_LIMIT 189

Default power limit in milliwatts (limit which device boots with).

#define NVML_FI_DEV_POWER_CURRENT_LIMIT 190

Limit currently enforced in milliwatts (This includes other limits set elsewhere. E.g. Out-of-band).

#define NVML_FI_DEV_ENERGY 191

Total energy consumption (in mJ) since the driver was last reloaded. Same as NVML_FI_DEV_TOTAL_ENERGY_CONSUMPTION for the GPU.

#define NVML_FI_DEV_POWER_REQUESTED_LIMIT 192

Power limit requested by NVML or any other userspace client.

#define NVML_FI_DEV_TEMPERATURE_SHUTDOWN_TLIMIT 193

T.Limit temperature after which GPU may shut down for HW protection.

GPU T.Limit temperature thresholds in degree Celsius

These fields are supported on Ada and later architectures and supersedes nvmlDeviceGetTemperatureThreshold.

#define NVML_FI_DEV_TEMPERATURE_SLOWDOWN_TLIMIT 194

T.Limit temperature after which GPU may begin HW slowdown.

#define NVML_FI_DEV_TEMPERATURE_MEM_MAX_TLIMIT 195

T.Limit temperature after which GPU may begin SW slowdown due to memory temperature.

#define NVML_FI_DEV_TEMPERATURE_GPU_MAX_TLIMIT 196

T.Limit temperature after which GPU may be throttled below base clock.

#define NVML_FI_DEV_IS_MIG_MODE_INDEPENDENT_MIG_QUERY_CAPABLE 199

MIG mode independent, MIG query capable device. 1=yes. 0=no.

#define NVML_FI_MAX 200

One greater than the largest field ID defined above.

4.4. Unit Structs

struct nvmlHwbcEntry_t

struct nvmlLedState_t

struct nvmlUnitInfo_t

struct nvmlPSUInfo_t

struct nvmlUnitFanInfo_t

struct nvmlUnitFanSpeeds_t

enum nvmlFanState_t

Fan state enum.

Values

NVML_FAN_NORMAL = 0
Fan is working properly.
NVML_FAN_FAILED = 1
Fan has failed.

enum nvmlLedColor_t

Led color enum.

Values

NVML_LED_COLOR_GREEN = 0
GREEN, indicates good health.
NVML_LED_COLOR_AMBER = 1
AMBER, indicates problem.

4.5. Accounting Statistics

Set of APIs designed to provide per process information about usage of GPU.

- All accounting statistics and accounting mode live in nvidia driver and reset to default (Disabled) when driver unloads. It is advised to run with persistence mode enabled.

- Enabling accounting mode has no negative impact on the GPU performance.

struct nvmlAccountingStats_t

nvmlReturn_t nvmlDeviceGetAccountingMode (nvmlDevice_t device, nvmlEnableState_t *mode)

Parameters

device
The identifier of the target device

mode
Reference in which to return the current accounting mode

Returns

- NVML_SUCCESS if the mode has been successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or mode are NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Queries the state of per process accounting mode.

For Kepler or newer fully supported devices.

See nvmlDeviceGetAccountingStats for more details. See nvmlDeviceSetAccountingMode

nvmlReturn_t nvmlDeviceGetAccountingStats (nvmlDevice_t device, unsigned int pid, nvmlAccountingStats_t *stats)

Parameters

device
The identifier of the target device

pid
Process Id of the target process to query stats for

stats
Reference in which to return the process's accounting stats

Returns

- NVML_SUCCESS if stats have been successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or stats are NULL
- NVML_ERROR_NOT_FOUND if process stats were not found
- NVML_ERROR_NOT_SUPPORTED if device doesn't support this feature or accounting mode is disabled or on vGPU host.
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Queries process's accounting stats.

For Kepler or newer fully supported devices.

Accounting stats capture GPU utilization and other statistics across the lifetime of a process. Accounting stats can be queried during life time of the process and after its termination. The time field in nvmlAccountingStats_t is reported as 0 during the lifetime of the process and updated to actual running time after its termination. Accounting stats are kept in a circular buffer, newly created processes overwrite information about old processes.

See nvmlAccountingStats_t for description of each returned metric. List of processes that can be queried can be retrieved from nvmlDeviceGetAccountingPids.

- Accounting Mode needs to be on. See nvmlDeviceGetAccountingMode.
- Only compute and graphics applications stats can be queried. Monitoring applications stats can't be queried since they don't contribute to GPU utilization.
- In case of pid collision stats of only the latest process (that terminated last) will be reported

See also:

nvmlDeviceGetAccountingBufferSize

nvmlReturn_t nvmlDeviceGetAccountingPids (nvmlDevice_t device, unsigned int *count, unsigned int *pids)

Parameters

device
The identifier of the target device

count
Reference in which to provide the pids array size, and to return the number of elements ready to be queried

pids
Reference in which to return list of process ids

Returns

- NVML_SUCCESS if pids were successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or count is NULL
- NVML_ERROR_NOT_SUPPORTED if device doesn't support this feature or accounting mode is disabled or on vGPU host.
- NVML_ERROR_INSUFFICIENT_SIZE if count is too small (count is set to expected value)
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Queries list of processes that can be queried for accounting stats. The list of processes returned can be in running or terminated state.

For Kepler or newer fully supported devices.

To just query the number of processes ready to be queried, call this function with *count = 0 and pids=NULL. The return code will be NVML_ERROR_INSUFFICIENT_SIZE, or NVML_SUCCESS if list is empty.

For more details see nvmlDeviceGetAccountingStats.

In case of PID collision some processes might not be accessible before the circular buffer is full.

See also:

nvmlDeviceGetAccountingBufferSize

nvmlReturn_t nvmlDeviceGetAccountingBufferSize (nvmlDevice_t device, unsigned int *bufferSize)

Parameters

device
The identifier of the target device

bufferSize
Reference in which to provide the size (in number of elements) of the circular buffer for accounting stats.

Returns

- NVML_SUCCESS if buffer size was successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or bufferSize is NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature or accounting mode is disabled
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Returns the number of processes that the circular buffer with accounting pids can hold.

For Kepler or newer fully supported devices.

This is the maximum number of processes that accounting information will be stored for before information about oldest processes will get overwritten by information about new processes.

See also:

nvmlDeviceGetAccountingStats

nvmlDeviceGetAccountingPids

nvmlReturn_t nvmlDeviceSetAccountingMode (nvmlDevice_t device, nvmlEnableState_t mode)

Parameters

device
The identifier of the target device
mode
The target accounting mode

Returns

- NVML_SUCCESS if the new mode has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device or mode are invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature

- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Enables or disables per process accounting.

For Kepler or newer fully supported devices. Requires root/admin permissions.

- This setting is not persistent and will default to disabled after driver unloads. Enable persistence mode to be sure the setting doesn't switch off to disabled.
- Enabling accounting mode has no negative impact on the GPU performance.
- Disabling accounting clears all accounting pids information.
- On MIG-enabled GPUs, accounting mode would be set to DISABLED and changing it is not supported.

See nvmlDeviceGetAccountingMode See nvmlDeviceGetAccountingStats See nvmlDeviceClearAccountingPids

nvmlReturn_t nvmlDeviceClearAccountingPids (nvmlDevice_t device)

Parameters

device
The identifier of the target device

Returns

- NVML_SUCCESS if accounting information has been cleared
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device are invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Clears accounting information about all processes that have already terminated.

For Kepler or newer fully supported devices. Requires root/admin permissions.

See nvmlDeviceGetAccountingMode See nvmlDeviceGetAccountingStats See nvmlDeviceSetAccountingMode

4.6. Encoder Structs

struct nvmlEncoderSessionInfo_t

enum nvmlEncoderType_t

Represents type of encoder for capacity can be queried

Values

NVML_ENCODER_QUERY_H264 = 0x00
H264 encoder.
NVML_ENCODER_QUERY_HEVC = 0x01
HEVC encoder.
NVML_ENCODER_QUERY_AV1 = 0x02
AV1 encoder.
NVML_ENCODER_QUERY_UNKNOWN = 0xFF
Unknown encoder.

4.7. Frame Buffer Capture Structures

struct nvmlFBCStats_t

struct nvmlFBCSessionInfo_t

enum nvmlFBCSessionType_t

Represents frame buffer capture session type

Values

NVML_FBC_SESSION_TYPE_UNKNOWN = 0
Unknown.
NVML_FBC_SESSION_TYPE_TOSYS
ToSys.

NVML_FBC_SESSION_TYPE_CUDA
Cuda.
NVML_FBC_SESSION_TYPE_VID
Vid.
NVML_FBC_SESSION_TYPE_HWENC
HEnc.

#define NVML_NVFBC_SESSION_FLAG_DIFFMAP_ENABLED 0x00000001

Bit specifying differential map state.

#define NVML_NVFBC_SESSION_FLAG_CLASSIFICATIONMAP_ENABLED 0x00000002

Bit specifying classification map state.

#define NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_NO_WAIT 0x00000004

Bit specifying if capture was requested as non-blocking call.

#define NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_INFINITE 0x00000008

Bit specifying if capture was requested as blocking call.

#define NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_TIMEOUT 0x00000010

Bit specifying if capture was requested as blocking call with timeout period.

4.8. definitions related to the drain state

enum nvmlDetachGpuState_t

Is the GPU device to be removed from the kernel by nvmlDeviceRemoveGpu()

Values

NVML_DETACH_GPU_KEEP = 0
NVML_DETACH_GPU_REMOVE

enum nvmlPcieLinkState_t

Parent bridge PCIe link state requested by nvmlDeviceRemoveGpu()

Values

NVML_PCIE_LINK_KEEP = 0
NVML_PCIE_LINK_SHUT_DOWN

4.9. /nvmlDevice definitions related to Confidential Computing

struct nvmlSystemConfComputeSettings_v1_t

struct nvmlConfComputeMemSizeInfo_t

#define NVML_CC_SYSTEM_CPU_CAPS_NONE 0

Confidential Compute CPU Capabilities values

#define NVML_CC_SYSTEM_GPUS_CC_NOT_CAPABLE 0

Confidenial Compute GPU Capabilities values

#define NVML_CC_SYSTEM_DEVTOOLS_MODE_OFF 0

Confidential Compute DevTools Mode values

#define NVML_CC_SYSTEM_ENVIRONMENT_UNAVAILABLE 0

Confidential Compute Environment values

#define NVML_CC_SYSTEM_FEATURE_DISABLED 0

Confidential Compute Feature Status values

#define NVML_CC_SYSTEM_MULTIGPU_NONE 0

Confidential Compute Multigpu mode values

#define NVML_CC_ACCEPTING_CLIENT_REQUESTS_FALSE 0

Confidential Compute GPUs/System Ready State values

#define NVML_GPU_CERT_CHAIN_SIZE 0x1000

GPU Certificate Details

#define NVML_CC_GPU_CEC_NONCE_SIZE 0x20

GPU Attestation Report

4.10. Initialization and Cleanup

This chapter describes the methods that handle NVML initialization and cleanup. It is the user's responsibility to call nvmlInit_v2() before calling any other methods, and nvmlShutdown() once NVML is no longer being used.

nvmlReturn_t nvmlInit_v2 (void)

Returns

- NVML_SUCCESS if NVML has been properly initialized
- NVML_ERROR_DRIVER_NOT_LOADED if NVIDIA driver is not running
- NVML_ERROR_NO_PERMISSION if NVML does not have permission to talk to the driver
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Initialize NVML, but don't initialize any GPUs yet.

- nvmlInit_v3 introduces a "flags" argument, that allows passing boolean values modifying the behaviour of nvmlInit().
- In NVML 5.319 new nvmlInit_v2 has replaced nvmlInit"_v1" (default in NVML 4.304 and older) that did initialize all GPU devices in the system.

This allows NVML to communicate with a GPU when other GPUs in the system are unstable or in a bad state. When using this API, GPUs are discovered and initialized in nvmlDeviceGetHandleBy* functions instead.

To contrast nvmlInit_v2 with nvmlInit"_v1", NVML 4.304 nvmlInit"_v1" will fail when any detected GPU is in a bad or unstable state.

For all products.

This method, should be called once before invoking any other methods in the library. A reference count of the number of initializations is maintained. Shutdown only occurs when the reference count reaches zero.

nvmlReturn_t nvmlInitWithFlags (unsigned int flags)

Parameters

flags
behaviour modifier flags

Returns

- NVML_SUCCESS if NVML has been properly initialized
- NVML_ERROR_DRIVER_NOT_LOADED if NVIDIA driver is not running
- NVML_ERROR_NO_PERMISSION if NVML does not have permission to talk to the driver
- NVML_ERROR_UNKNOWN on any unexpected error

Description

nvmlInitWithFlags is a variant of nvmlInit(), that allows passing a set of boolean values modifying the behaviour of nvmlInit(). Other than the "flags" parameter it is completely similar to nvmlInit_v2.

For all products.

nvmlReturn_t nvmlShutdown (void)

Returns

- NVML_SUCCESS if NVML has been properly shut down
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Shut down NVML by releasing all GPU resources previously allocated with nvmlInit_v2().

For all products.

This method should be called after NVML work is done, once for each call to nvmlInit_v2() A reference count of the number of initializations is maintained. Shutdown only occurs when the reference count reaches zero. For backwards compatibility, no error is reported if nvmlShutdown() is called more times than nvmlInit().

#define NVML_INIT_FLAG_NO_GPUS 1

Don't fail nvmlInit() when no GPUs are found.

#define NVML_INIT_FLAG_NO_ATTACH 2

Don't attach GPUs.

4.11. Error reporting

This chapter describes helper functions for error reporting routines.

const DECLDIR char *nvmlErrorString (nvmlReturn_t result)

Parameters

result
NVML error code to convert

Returns

String representation of the error.

Description

Helper method for converting NVML error codes into readable strings.

For all products.

4.12. Constants

#define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE 16

Buffer size guaranteed to be large enough for nvmlDeviceGetInforomVersion and nvmlDeviceGetInforomImageVersion

#define NVML_DEVICE_UUID_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for storing GPU identifiers.

#define NVML_DEVICE_UUID_V2_BUFFER_SIZE 96

Buffer size guaranteed to be large enough for nvmlDeviceGetUUID

#define NVML_DEVICE_PART_NUMBER_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for nvmlDeviceGetBoardPartNumber

#define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for nvmlSystemGetDriverVersion

#define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for nvmlSystemGetNVMLVersion

#define NVML_DEVICE_NAME_BUFFER_SIZE 64

Buffer size guaranteed to be large enough for storing GPU device names.

#define NVML_DEVICE_NAME_V2_BUFFER_SIZE 96

Buffer size guaranteed to be large enough for nvmlDeviceGetName

#define NVML_DEVICE_SERIAL_BUFFER_SIZE 30

Buffer size guaranteed to be large enough for nvmlDeviceGetSerial

#define NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE 32

Buffer size guaranteed to be large enough for nvmlDeviceGetVbiosVersion

4.13. System Queries

This chapter describes the queries that NVML can perform against the local system. These queries are not device-specific.

nvmlReturn_t nvmlSystemGetDriverVersion (char *version, unsigned int length)

Parameters

version
Reference in which to return the version identifier

length
The maximum allowed length of the string returned in version

Returns

- NVML_SUCCESS if version has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if version is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small

Description

Retrieves the version of the system's graphics driver.

For all products.

The version identifier is an alphanumeric string. It will not exceed 80 characters in length (including the NULL terminator). See nvmlConstants::NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE.

nvmlReturn_t nvmlSystemGetNVMLVersion (char *version, unsigned int length)

Parameters

version
Reference in which to return the version identifier

length
The maximum allowed length of the string returned in version

Returns

- NVML_SUCCESS if version has been set
- NVML_ERROR_INVALID_ARGUMENT if version is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small

Description

Retrieves the version of the NVML library.

For all products.

The version identifier is an alphanumeric string. It will not exceed 80 characters in length (including the NULL terminator). See nvmlConstants::NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE.

nvmlReturn_t nvmlSystemGetCudaDriverVersion (int *cudaDriverVersion)

Parameters

cudaDriverVersion
Reference in which to return the version identifier

Returns

- NVML_SUCCESS if cudaDriverVersion has been set
- NVML_ERROR_INVALID_ARGUMENT if cudaDriverVersion is NULL

Description

Retrieves the version of the CUDA driver.

For all products.

The CUDA driver version returned will be retreived from the currently installed version of CUDA. If the cuda library is not found, this function will return a known supported version number.

nvmlReturn_t nvmlSystemGetCudaDriverVersion_v2 (int *cudaDriverVersion)

Parameters

cudaDriverVersion
Reference in which to return the version identifier

Returns

- NVML_SUCCESS if cudaDriverVersion has been set
- NVML_ERROR_INVALID_ARGUMENT if cudaDriverVersion is NULL
- NVML_ERROR_LIBRARY_NOT_FOUND if libcuda.so.1 or libcuda.dll is not found
- NVML_ERROR_FUNCTION_NOT_FOUND if cuDriverGetVersion() is not found in the shared library

Description

Retrieves the version of the CUDA driver from the shared library.

For all products.

The returned CUDA driver version by calling cuDriverGetVersion()

nvmlReturn_t nvmlSystemGetProcessName (unsigned int pid, char *name, unsigned int length)

Parameters

pid
The identifier of the process
name
Reference in which to return the process name
length
The maximum allowed length of the string returned in name

Returns

- NVML_SUCCESS if name has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if name is NULL or length is 0.
- NVML_ERROR_NOT_FOUND if process doesn't exists
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Gets name of the process with provided process id

For all products.

Returned process name is cropped to provided length. name string is encoded in ANSI.

nvmlReturn_t nvmlSystemGetHicVersion (unsigned int *hwbcCount, nvmlHwbcEntry_t *hwbcEntries)

Parameters

hwbcCount
Size of hwbcEntries array

hwbcEntries
Array holding information about hwbc

Returns

- NVML_SUCCESS if hwbcCount and hwbcEntries have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if either hwbcCount or hwbcEntries is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if hwbcCount indicates that the hwbcEntries array is too small

Description

Retrieves the IDs and firmware versions for any Host Interface Cards (HICs) in the system.

For S-class products.

The hwbcCount argument is expected to be set to the size of the input hwbcEntries array. The HIC must be connected to an S-class system for it to be reported by this function.

nvmlReturn_t nvmlSystemGetTopologyGpuSet (unsigned int cpuNumber, unsigned int *count, nvmlDevice_t *deviceArray)

Parameters

cpuNumber
The CPU number

count
When zero, is set to the number of matching GPUs such that deviceArray can be malloc'd. When non-zero, deviceArray will be filled with count number of device handles.

deviceArray
An array of device handles for GPUs found with affinity to cpuNumber

Returns

- NVML_SUCCESS if deviceArray or count (if initially zero) has been set
- NVML_ERROR_INVALID_ARGUMENT if cpuNumber, or count is invalid, or deviceArray is NULL with a non-zero count
- NVML_ERROR_NOT_SUPPORTED if the device or OS does not support this feature
- NVML_ERROR_UNKNOWN an error has occurred in underlying topology discovery

Description

Retrieve the set of GPUs that have a CPU affinity with the given CPU number For all products. Supported on Linux only.

#define NVML_CUDA_DRIVER_VERSION_MAJOR ((v)/1000)

Macros for converting the CUDA driver version number to Major and Minor version numbers.

4.14. Unit Queries

This chapter describes that queries that NVML can perform against each unit. For S-class systems only. In each case the device is identified with an nvmlUnit_t handle. This handle is obtained by calling nvmlUnitGetHandleByIndex().

nvmlReturn_t nvmlUnitGetCount (unsigned int *unitCount)

Parameters

unitCount
Reference in which to return the number of units

Returns

- NVML_SUCCESS if unitCount has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if unitCount is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the number of units in the system.

For S-class products.

nvmlReturn_t nvmlUnitGetHandleByIndex (unsigned int index, nvmlUnit_t *unit)

Parameters

index
The index of the target unit, >= 0 and < unitCount
unit
Reference in which to return the unit handle

Returns

- NVML_SUCCESS if unit has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if index is invalid or unit is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Acquire the handle for a particular unit, based on its index.

For S-class products.

Valid indices are derived from the unitCount returned by nvmlUnitGetCount(). For example, if unitCount is 2 the valid indices are 0 and 1, corresponding to UNIT 0 and UNIT 1.

The order in which NVML enumerates units has no guarantees of consistency between reboots.

nvmlReturn_t nvmlUnitGetUnitInfo (nvmlUnit_t unit, nvmlUnitInfo_t *info)

Parameters

unit
The identifier of the target unit

info
Reference in which to return the unit information

Returns

- NVML_SUCCESS if info has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if unit is invalid or info is NULL

Description

Retrieves the static information associated with a unit.

For S-class products.

See nvmlUnitInfo_t for details on available unit info.

nvmlReturn_t nvmlUnitGetLedState (nvmlUnit_t unit, nvmlLedState_t *state)

Parameters

unit
The identifier of the target unit

state
Reference in which to return the current LED state

Returns

- NVML_SUCCESS if state has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if unit is invalid or state is NULL
- NVML_ERROR_NOT_SUPPORTED if this is not an S-class product
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the LED state associated with this unit.

For S-class products.

See nvmlLedState_t for details on allowed states.

See also:

nvmlUnitSetLedState()

nvmlReturn_t nvmlUnitGetPsuInfo (nvmlUnit_t unit, nvmlPSUInfo_t *psu)

Parameters

unit
The identifier of the target unit

psu
Reference in which to return the PSU information

Returns

- NVML_SUCCESS if psu has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if unit is invalid or psu is NULL
- NVML_ERROR_NOT_SUPPORTED if this is not an S-class product
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the PSU stats for the unit.

For S-class products.

See nvmlPSUInfo_t for details on available PSU info.

nvmlReturn_t nvmlUnitGetTemperature (nvmlUnit_t unit, unsigned int type, unsigned int *temp)

Parameters

unit
The identifier of the target unit

type
The type of reading to take

temp
Reference in which to return the intake temperature

Returns

- NVML_SUCCESS if temp has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if unit or type is invalid or temp is NULL
- NVML_ERROR_NOT_SUPPORTED if this is not an S-class product
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the temperature readings for the unit, in degrees C.

For S-class products.

Depending on the product, readings may be available for intake (type=0), exhaust (type=1) and board (type=2).

nvmlReturn_t nvmlUnitGetFanSpeedInfo (nvmlUnit_t unit, nvmlUnitFanSpeeds_t *fanSpeeds)

Parameters

unit
The identifier of the target unit

fanSpeeds
Reference in which to return the fan speed information

Returns

- NVML_SUCCESS if fanSpeeds has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if unit is invalid or fanSpeeds is NULL

- NVML_ERROR_NOT_SUPPORTED if this is not an S-class product
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the fan speed readings for the unit.

For S-class products.

See nvmlUnitFanSpeeds_t for details on available fan speed info.

nvmlReturn_t nvmlUnitGetDevices (nvmlUnit_t unit, unsigned int *deviceCount, nvmlDevice_t *devices)

Parameters

unit
The identifier of the target unit

deviceCount
Reference in which to provide the devices array size, and to return the number of attached GPU devices

devices
Reference in which to return the references to the attached GPU devices

Returns

- NVML_SUCCESS if deviceCount and devices have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INSUFFICIENT_SIZE if deviceCount indicates that the devices array is too small
- NVML_ERROR_INVALID_ARGUMENT if unit is invalid, either of deviceCount or devices is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the set of GPU devices that are attached to the specified unit.

For S-class products.

The deviceCount argument is expected to be set to the size of the input devices array.

4.15. Device Queries

This chapter describes that queries that NVML can perform against each device. In each case the device is identified with an nvmlDevice_t handle. This handle is obtained by calling one of nvmlDeviceGetHandleByIndex_v2(), nvmlDeviceGetHandleBySerial(), nvmlDeviceGetHandleByPciBusId_v2(). or nvmlDeviceGetHandleByUUID().

CPU and Memory Affinity

nvmlReturn_t nvmlDeviceGetCount_v2 (unsigned int *deviceCount)

Parameters

deviceCount
Reference in which to return the number of accessible devices

Returns

- NVML_SUCCESS if deviceCount has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if deviceCount is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the number of compute devices in the system. A compute device is a single GPU.

For all products.

Note: New nvmlDeviceGetCount_v2 (default in NVML 5.319) returns count of all devices in the system even if nvmlDeviceGetHandleByIndex_v2 returns NVML_ERROR_NO_PERMISSION for such device. Update your code to handle this error, or use NVML 4.304 or older nvml header file. For backward binary compatibility reasons _v1 version of the API is still present in the shared library. Old _v1 version of nvmlDeviceGetCount doesn't count devices that NVML has no permission to talk to.

nvmlReturn_t nvmlDeviceGetAttributes_v2 (nvmlDevice_t device, nvmlDeviceAttributes_t *attributes)

Parameters

device
NVML device handle
attributes
Device attributes

Returns

- NVML_SUCCESS if device attributes were successfully retrieved
- NVML_ERROR_INVALID_ARGUMENT if device handle is invalid
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get attributes (engine counts etc.) for the given NVML device handle.

This API currently only supports MIG device handles.

For Ampere or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetHandleByIndex_v2 (unsigned int index, nvmlDevice_t *device)

Parameters

index
The index of the target GPU, >= 0 and < accessibleDevices
device
Reference in which to return the device handle

Returns

- NVML_SUCCESS if device has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if index is invalid or device is NULL

- NVML_ERROR_INSUFFICIENT_POWER if any attached devices have improperly attached external power cables
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to talk to this device
- NVML_ERROR_IRQ_ISSUE if NVIDIA kernel detected an interrupt issue with the attached GPUs
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Acquire the handle for a particular device, based on its index.

For all products.

Valid indices are derived from the accessibleDevices count returned by nvmlDeviceGetCount_v2(). For example, if accessibleDevices is 2 the valid indices are 0 and 1, corresponding to GPU 0 and GPU 1.

The order in which NVML enumerates devices has no guarantees of consistency between reboots. For that reason it is recommended that devices be looked up by their PCI ids or UUID. See nvmlDeviceGetHandleByUUID() and nvmlDeviceGetHandleByPciBusId_v2().

Note: The NVML index may not correlate with other APIs, such as the CUDA device index.

Starting from NVML 5, this API causes NVML to initialize the target GPU NVML may initialize additional GPUs if:

- The target GPU is an SLI slave

Note: New nvmlDeviceGetCount_v2 (default in NVML 5.319) returns count of all devices in the system even if nvmlDeviceGetHandleByIndex_v2 returns NVML_ERROR_NO_PERMISSION for such device. Update your code to handle this error, or use NVML 4.304 or older nvml header file. For backward binary compatibility reasons _v1 version of the API is still present in the shared library. Old _v1 version of nvmlDeviceGetCount doesn't count devices that NVML has no permission to talk to.

This means that nvmlDeviceGetHandleByIndex_v2 and _v1 can return different devices for the same index. If you don't touch macros that map old (_v1) versions to _v2 versions at the top of the file you don't need to worry about that.

See also:

nvmlDeviceGetIndex

nvmlDeviceGetCount

nvmlReturn_t nvmlDeviceGetHandleBySerial (const char *serial, nvmlDevice_t *device)

Parameters

serial
The board serial number of the target GPU

device
Reference in which to return the device handle

Returns

- NVML_SUCCESS if device has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if serial is invalid, device is NULL or more than one device has the same serial (dual GPU boards)
- NVML_ERROR_NOT_FOUND if serial does not match a valid device on the system
- NVML_ERROR_INSUFFICIENT_POWER if any attached devices have improperly attached external power cables
- NVML_ERROR_IRQ_ISSUE if NVIDIA kernel detected an interrupt issue with the attached GPUs
- NVML_ERROR_GPU_IS_LOST if any GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Acquire the handle for a particular device, based on its board serial number.

For Fermi or newer fully supported devices.

This number corresponds to the value printed directly on the board, and to the value returned by nvmlDeviceGetSerial().

Deprecated Since more than one GPU can exist on a single board this function is deprecated in favor of nvmlDeviceGetHandleByUUID. For dual GPU boards this function will return NVML_ERROR_INVALID_ARGUMENT.

Starting from NVML 5, this API causes NVML to initialize the target GPU NVML may initialize additional GPUs as it searches for the target GPU

See also:

nvmlDeviceGetSerial

nvmlDeviceGetHandleByUUID

nvmlReturn_t nvmlDeviceGetHandleByUUID (const char *uuid, nvmlDevice_t *device)

Parameters

uuid
The UUID of the target GPU or MIG instance

device
Reference in which to return the device handle or MIG device handle

Returns

- NVML_SUCCESS if device has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if uuid is invalid or device is null
- NVML_ERROR_NOT_FOUND if uuid does not match a valid device on the system
- NVML_ERROR_INSUFFICIENT_POWER if any attached devices have improperly attached external power cables
- NVML_ERROR_IRQ_ISSUE if NVIDIA kernel detected an interrupt issue with the attached GPUs
- NVML_ERROR_GPU_IS_LOST if any GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Acquire the handle for a particular device, based on its globally unique immutable UUID associated with each device.

For all products.

Starting from NVML 5, this API causes NVML to initialize the target GPU NVML may initialize additional GPUs as it searches for the target GPU

See also:

nvmlDeviceGetUUID

nvmlReturn_t nvmlDeviceGetHandleByPciBusId_v2 (const char *pciBusId, nvmlDevice_t *device)

Parameters

pciBusId
The PCI bus id of the target GPU

device
Reference in which to return the device handle

Returns

- NVML_SUCCESS if device has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if pciBusId is invalid or device is NULL
- NVML_ERROR_NOT_FOUND if pciBusId does not match a valid device on the system
- NVML_ERROR_INSUFFICIENT_POWER if the attached device has improperly attached external power cables
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to talk to this device
- NVML_ERROR_IRQ_ISSUE if NVIDIA kernel detected an interrupt issue with the attached GPUs
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Acquire the handle for a particular device, based on its PCI bus id.

For all products.

This value corresponds to the nvmlPciInfo_t::busId returned by nvmlDeviceGetPciInfo_v3().

Starting from NVML 5, this API causes NVML to initialize the target GPU NVML may initialize additional GPUs if:

- The target GPU is an SLI slave

NVML 4.304 and older version of nvmlDeviceGetHandleByPciBusId"_v1" returns NVML_ERROR_NOT_FOUND instead of NVML_ERROR_NO_PERMISSION.

nvmlReturn_t nvmlDeviceGetName (nvmlDevice_t device, char *name, unsigned int length)

Parameters

device
The identifier of the target device

name
Reference in which to return the product name

length
The maximum allowed length of the string returned in name

Returns

- NVML_SUCCESS if name has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or name is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the name of this device.

For all products.

The name is an alphanumeric string that denotes a particular product, e.g. Tesla C2070. It will not exceed 96 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_NAME_V2_BUFFER_SIZE.

When used with MIG device handles the API returns MIG device names which can be used to identify devices based on their attributes.

nvmlReturn_t nvmlDeviceGetBrand (nvmlDevice_t device, nvmlBrandType_t *type)

Parameters

device
The identifier of the target device

type
Reference in which to return the product brand type

Returns

- NVML_SUCCESS if name has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or type is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the brand of this device.

For all products.

The type is a member of nvmlBrandType_t defined above.

nvmlReturn_t nvmlDeviceGetIndex (nvmlDevice_t device, unsigned int *index)

Parameters

device
The identifier of the target device

index
Reference in which to return the NVML index of the device

Returns

- NVML_SUCCESS if index has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or index is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the NVML index of this device.

For all products.

Valid indices are derived from the accessibleDevices count returned by nvmlDeviceGetCount_v2(). For example, if accessibleDevices is 2 the valid indices are 0 and 1, corresponding to GPU 0 and GPU 1.

The order in which NVML enumerates devices has no guarantees of consistency between reboots. For that reason it is recommended that devices be looked up by their PCI ids or GPU UUID. See nvmlDeviceGetHandleByPciBusId_v2() and nvmlDeviceGetHandleByUUID().

When used with MIG device handles this API returns indices that can be passed to nvmlDeviceGetMigDeviceHandleByIndex to retrieve an identical handle. MIG device indices are unique within a device.

Note: The NVML index may not correlate with other APIs, such as the CUDA device index.

See also:

nvmlDeviceGetHandleByIndex()

nvmlDeviceGetCount()

nvmlReturn_t nvmlDeviceGetSerial (nvmlDevice_t device, char *serial, unsigned int length)

Parameters

device
The identifier of the target device
serial
Reference in which to return the board/module serial number
length
The maximum allowed length of the string returned in serial

Returns

- NVML_SUCCESS if serial has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or serial is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the globally unique board serial number associated with this device's board.

For all products with an inforom.

The serial number is an alphanumeric string that will not exceed 30 characters (including the NULL terminator). This number matches the serial number tag that is physically attached to the board. See nvmlConstants::NVML_DEVICE_SERIAL_BUFFER_SIZE.

nvmlReturn_t nvmlDeviceGetC2cModeInfoV (nvmlDevice_t device, nvmlC2cModeInfo_v1_t *c2cModeInfo)

Parameters

device
The identifier of the target device
c2cModeInfo
Output struct containing the device's C2C Mode info

Returns

- NVML_SUCCESS if C2C Mode Infor query is successful
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or serial is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the Device's C2C Mode information

nvmlReturn_t nvmlDeviceGetTopologyCommonAncestor (nvmlDevice_t device1, nvmlDevice_t device2, nvmlGpuTopologyLevel_t *pathInfo)

Parameters

device1
The identifier of the first device
device2
The identifier of the second device
pathInfo
A nvmlGpuTopologyLevel_t that gives the path type

Returns

- NVML_SUCCESS if pathInfo has been set
- NVML_ERROR_INVALID_ARGUMENT if device1, or device2 is invalid, or pathInfo is NULL
- NVML_ERROR_NOT_SUPPORTED if the device or OS does not support this feature
- NVML_ERROR_UNKNOWN an error has occurred in underlying topology discovery

Description

Retrieve the common ancestor for two devices For all products. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetTopologyNearestGpus (nvmlDevice_t device, nvmlGpuTopologyLevel_t level, unsigned int *count, nvmlDevice_t *deviceArray)

Parameters

device
The identifier of the first device

level
The nvmlGpuTopologyLevel_t level to search for other GPUs

count
When zero, is set to the number of matching GPUs such that deviceArray can be malloc'd. When non-zero, deviceArray will be filled with count number of device handles.

deviceArray
An array of device handles for GPUs found at level

Returns

- NVML_SUCCESS if deviceArray or count (if initially zero) has been set
- NVML_ERROR_INVALID_ARGUMENT if device, level, or count is invalid, or deviceArray is NULL with a non-zero count
- NVML_ERROR_NOT_SUPPORTED if the device or OS does not support this feature
- NVML_ERROR_UNKNOWN an error has occurred in underlying topology discovery

Description

Retrieve the set of GPUs that are nearest to a given device at a specific interconnectivity level For all products. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetP2PStatus (nvmlDevice_t device1, nvmlDevice_t device2, nvmlGpuP2PCapsIndex_t p2pIndex, nvmlGpuP2PStatus_t *p2pStatus)

Parameters

device1
The first device
device2
The second device
p2pIndex
p2p Capability Index being looked for between device1 and device2
p2pStatus
Reference in which to return the status of the p2pIndex between device1 and device2

Returns

- NVML_SUCCESS if p2pStatus has been populated
- NVML_ERROR_INVALID_ARGUMENT if device1 or device2 or p2pIndex is invalid or p2pStatus is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the status for a given p2p capability index between a given pair of GPU

nvmlReturn_t nvmlDeviceGetUUID (nvmlDevice_t device, char *uuid, unsigned int length)

Parameters

device
The identifier of the target device
uuid
Reference in which to return the GPU UUID

length
The maximum allowed length of the string returned in uuid

Returns

- NVML_SUCCESS if uuid has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or uuid is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the globally unique immutable UUID associated with this device, as a 5 part hexadecimal string, that augments the immutable, board serial identifier.

For all products.

The UUID is a globally unique identifier. It is the only available identifier for pre-Fermi-architecture products. It does NOT correspond to any identifier printed on the board. It will not exceed 96 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_UUID_V2_BUFFER_SIZE.

When used with MIG device handles the API returns globally unique UUIDs which can be used to identify MIG devices across both GPU and MIG devices. UUIDs are immutable for the lifetime of a MIG device.

nvmlReturn_t nvmlDeviceGetMinorNumber (nvmlDevice_t device, unsigned int *minorNumber)

Parameters

device
The identifier of the target device
minorNumber
Reference in which to return the minor number for the device

Returns

- NVML_SUCCESS if the minor number is successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if device is invalid or minorNumber is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves minor number for the device. The minor number for the device is such that the Nvidia device node file for each GPU will have the form /dev/nvidia[minor number].

For all products. Supported only for Linux

nvmlReturn_t nvmlDeviceGetBoardPartNumber (nvmlDevice_t device, char *partNumber, unsigned int length)

Parameters

device
Identifier of the target device
partNumber
Reference to the buffer to return
length
Length of the buffer reference

Returns

- NVML_SUCCESS if partNumber has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_SUPPORTED if the needed VBIOS fields have not been filled
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or serial is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the the device board part number which is programmed into the board's InfoROM

For all products.

nvmlReturn_t nvmlDeviceGetInforomVersion (nvmlDevice_t device, nvmlInforomObject_t object, char *version, unsigned int length)

Parameters

device
The identifier of the target device
object
The target infoROM object
version
Reference in which to return the infoROM version
length
The maximum allowed length of the string returned in version

Returns

- NVML_SUCCESS if version has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if version is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small
- NVML_ERROR_NOT_SUPPORTED if the device does not have an infoROM
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the version information for the device's infoROM object.

For all products with an inforom.

Fermi and higher parts have non-volatile on-board memory for persisting device info, such as aggregate ECC counts. The version of the data structures in this memory may change from time to time. It will not exceed 16 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE.

See nvmlInforomObject_t for details on the available infoROM objects.

See also:

nvmlDeviceGetInforomImageVersion

nvmlReturn_t nvmlDeviceGetInforomImageVersion (nvmlDevice_t device, char *version, unsigned int length)

Parameters

device
The identifier of the target device
version
Reference in which to return the infoROM image version
length
The maximum allowed length of the string returned in version

Returns

- NVML_SUCCESS if version has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if version is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small
- NVML_ERROR_NOT_SUPPORTED if the device does not have an infoROM
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the global infoROM image version

For all products with an inforom.

Image version just like VBIOS version uniquely describes the exact version of the infoROM flashed on the board in contrast to infoROM object version which is only an indicator of supported features. Version string will not exceed 16 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE.

See also:

nvmlDeviceGetInforomVersion

nvmlReturn_t nvmlDeviceGetInforomConfigurationChecksum (nvmlDevice_t device, unsigned int *checksum)

Parameters

device
The identifier of the target device

checksum
Reference in which to return the infoROM configuration checksum

Returns

- NVML_SUCCESS if checksum has been set
- NVML_ERROR_CORRUPTED_INFOROM if the device's checksum couldn't be retrieved due to infoROM corruption
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if checksum is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the checksum of the configuration stored in the device's infoROM.

For all products with an inforom.

Can be used to make sure that two GPUs have the exact same configuration. Current checksum takes into account configuration stored in PWR and ECC infoROM objects. Checksum can change between driver releases or when user changes configuration (e.g. disable/enable ECC)

nvmlReturn_t nvmlDeviceValidateInforom (nvmlDevice_t device)

Parameters

device
The identifier of the target device

Returns

- NVML_SUCCESS if infoROM is not corrupted
- NVML_ERROR_CORRUPTED_INFOROM if the device's infoROM is corrupted
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Reads the infoROM from the flash and verifies the checksums.

For all products with an inforom.

nvmlReturn_t nvmlDeviceGetLastBBXFlushTime (nvmlDevice_t device, unsigned long long *timestamp, unsignedlong *durationUs)

Parameters

device
The identifier of the target device

timestamp
The start timestamp of the last BBX Flush

durationUs
The duration (us) of the last BBX Flush

Returns

- NVML_SUCCESS if timestamp and durationUs are successfully retrieved
- NVML_ERROR_NOT_READY if the BBX object has not been flushed yet
- NVML_ERROR_NOT_SUPPORTED if the device does not have an infoROM
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the timestamp and the duration of the last flush of the BBX (blackbox) infoROM object during the current run.

For all products with an inforom.

See also:

nvmlDeviceGetInforomVersion

nvmlReturn_t nvmlDeviceGetDisplayMode (nvmlDevice_t device, nvmlEnableState_t *display)

Parameters

device
The identifier of the target device

display
Reference in which to return the display mode

Returns

- NVML_SUCCESS if display has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or display is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the display mode for the device.

For all products.

This method indicates whether a physical display (e.g. monitor) is currently connected to any of the device's connectors.

See nvmlEnableState_t for details on allowed modes.

nvmlReturn_t nvmlDeviceGetDisplayActive (nvmlDevice_t device, nvmlEnableState_t *isActive)

Parameters

device
The identifier of the target device

isActive
Reference in which to return the display active state

Returns

- NVML_SUCCESS if isActive has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or isActive is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the display active state for the device.

For all products.

This method indicates whether a display is initialized on the device. For example whether X Server is attached to this device and has allocated memory for the screen.

Display can be active even when no monitor is physically attached.

See nvmlEnableState_t for details on allowed modes.

nvmlReturn_t nvmlDeviceGetPersistenceMode (nvmlDevice_t device, nvmlEnableState_t *mode)

Parameters

device
The identifier of the target device

mode
Reference in which to return the current driver persistence mode

Returns

- NVML_SUCCESS if mode has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or mode is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the persistence mode associated with this device.

For all products. For Linux only.

When driver persistence mode is enabled the driver software state is not torn down when the last client disconnects. By default this feature is disabled.

See nvmlEnableState_t for details on allowed modes.

See also:

nvmlDeviceSetPersistenceMode()

nvmlReturn_t nvmlDeviceGetPciInfoExt (nvmlDevice_t device, nvmlPciInfoExt_t *pci)

Parameters

device
The identifier of the target device

pci
Reference in which to return the PCI info

Returns

- NVML_SUCCESS if pci has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or pci is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves PCI attributes of this device.

For all products.

See nvmlPciInfoExt_t for details on the available PCI info.

nvmlReturn_t nvmlDeviceGetPciInfo_v3 (nvmlDevice_t device, nvmlPciInfo_t *pci)

Parameters

device
The identifier of the target device

pci
Reference in which to return the PCI info

Returns

- NVML_SUCCESS if pci has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or pci is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the PCI attributes of this device.

For all products.

See nvmlPciInfo_t for details on the available PCI info.

nvmlReturn_t nvmlDeviceGetMaxPcieLinkGeneration (nvmlDevice_t device, unsigned int *maxLinkGen)

Parameters

device
The identifier of the target device
maxLinkGen
Reference in which to return the max PCIe link generation

Returns

- NVML_SUCCESS if maxLinkGen has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or maxLinkGen is null
- NVML_ERROR_NOT_SUPPORTED if PCIe link information is not available
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the maximum PCIe link generation possible with this device and system

I.E. for a generation 2 PCIe device attached to a generation 1 PCIe bus the max link generation this function will report is generation 1.

For Fermi or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetGpuMaxPcieLinkGeneration (nvmlDevice_t device, unsigned int *maxLinkGenDevice)

Parameters

device
The identifier of the target device
maxLinkGenDevice
Reference in which to return the max PCIe link generation

Returns

- NVML_SUCCESS if maxLinkGenDevice has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or maxLinkGenDevice is null
- NVML_ERROR_NOT_SUPPORTED if PCIe link information is not available
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the maximum PCIe link generation supported by this device

For Fermi or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetMaxPcieLinkWidth (nvmlDevice_t device, unsigned int *maxLinkWidth)

Parameters

device
The identifier of the target device
maxLinkWidth
Reference in which to return the max PCIe link generation

Returns

- NVML_SUCCESS if maxLinkWidth has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if device is invalid or maxLinkWidth is null
- NVML_ERROR_NOT_SUPPORTED if PCIe link information is not available
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the maximum PCIe link width possible with this device and system

I.E. for a device with a 16x PCIe bus width attached to a 8x PCIe system bus this function will report a max link width of 8.

For Fermi or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetCurrPcieLinkGeneration (nvmlDevice_t device, unsigned int *currLinkGen)

Parameters

device
The identifier of the target device

currLinkGen
Reference in which to return the current PCIe link generation

Returns

- NVML_SUCCESS if currLinkGen has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or currLinkGen is null
- NVML_ERROR_NOT_SUPPORTED if PCIe link information is not available
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current PCIe link generation

For Fermi or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetCurrPcieLinkWidth (nvmlDevice_t device, unsigned int *currLinkWidth)

Parameters

device
The identifier of the target device

currLinkWidth
Reference in which to return the current PCIe link generation

Returns

- NVML_SUCCESS if currLinkWidth has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or currLinkWidth is null
- NVML_ERROR_NOT_SUPPORTED if PCIe link information is not available
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current PCIe link width

For Fermi or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetPcieThroughput (nvmlDevice_t device, nvmlPcieUtilCounter_t counter, unsigned int *value)

Parameters

device
The identifier of the target device

counter
The specific counter that should be queried nvmlPcieUtilCounter_t

value
Reference in which to return throughput in KB/s

Returns

- NVML_SUCCESS if value has been set

- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device or counter is invalid, or value is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve PCIe utilization information. This function is querying a byte counter over a 20ms interval and thus is the PCIe throughput over that interval.

For Maxwell or newer fully supported devices.

This method is not supported in virtual machines running virtual GPU (vGPU).

nvmlReturn_t nvmlDeviceGetPcieReplayCounter (nvmlDevice_t device, unsigned int *value)

Parameters

device
The identifier of the target device

value
Reference in which to return the counter's value

Returns

- NVML_SUCCESS if value has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or value is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the PCIe replay counter.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetClockInfo (nvmlDevice_t device, nvmlClockType_t type, unsigned int *clock)

Parameters

device
The identifier of the target device
type
Identify which clock domain to query
clock
Reference in which to return the clock speed in MHz

Returns

- NVML_SUCCESS if clock has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or clock is NULL
- NVML_ERROR_NOT_SUPPORTED if the device cannot report the specified clock
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current clock speeds for the device.

For Fermi or newer fully supported devices.

See nvmlClockType_t for details on available clock information.

nvmlReturn_t nvmlDeviceGetMaxClockInfo (nvmlDevice_t device, nvmlClockType_t type, unsigned int *clock)

Parameters

device
The identifier of the target device
type
Identify which clock domain to query
clock
Reference in which to return the clock speed in MHz

Returns

- NVML_SUCCESS if clock has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or clock is NULL
- NVML_ERROR_NOT_SUPPORTED if the device cannot report the specified clock
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the maximum clock speeds for the device.

For Fermi or newer fully supported devices.

See nvmlClockType_t for details on available clock information.

On GPUs from Fermi family current P0 clocks (reported by nvmlDeviceGetClockInfo) can differ from max clocks by few MHz.

nvmlReturn_t nvmlDeviceGetGpcClkVfOffset (nvmlDevice_t device, int *offset)

Parameters

device
The identifier of the target device

offset
The retrieved GPCCLK VF offset value

Returns

- NVML_SUCCESS if offset has been successfully queried
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or offset is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the GPCCLK VF offset value

nvmlReturn_t nvmlDeviceGetApplicationsClock (nvmlDevice_t device, nvmlClockType_t clockType, unsigned int *clockMHz)

Parameters

device
The identifier of the target device
clockType
Identify which clock domain to query
clockMHz
Reference in which to return the clock in MHz

Returns

- NVML_SUCCESS if clockMHz has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or clockMHz is NULL or clockType is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current setting of a clock that applications will use unless an overspec situation occurs. Can be changed using nvmlDeviceSetApplicationsClocks.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetDefaultApplicationsClock (nvmlDevice_t device, nvmlClockType_t clockType, unsigned int *clockMHz)

Parameters

device
The identifier of the target device
clockType
Identify which clock domain to query

clockMHz
Reference in which to return the default clock in MHz

Returns

- NVML_SUCCESS if clockMHz has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or clockMHz is NULL or clockType is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the default applications clock that GPU boots with or defaults to after nvmlDeviceResetApplicationsClocks call.

For Kepler or newer fully supported devices.

See also:

nvmlDeviceGetApplicationsClock

nvmlReturn_t nvmlDeviceGetClock (nvmlDevice_t device, nvmlClockType_t clockType, nvmlClockId_t clockId, unsigned int *clockMHz)

Parameters

device
The identifier of the target device
clockType
Identify which clock domain to query
clockId
Identify which clock in the domain to query
clockMHz
Reference in which to return the clock in MHz

Returns

- NVML_SUCCESS if clockMHz has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if device is invalid or clockMHz is NULL or clockType is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the clock speed for the clock specified by the clock type and clock ID.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetMaxCustomerBoostClock (nvmlDevice_t device, nvmlClockType_t clockType, unsigned int *clockMHz)

Parameters

device
The identifier of the target device
clockType
Identify which clock domain to query
clockMHz
Reference in which to return the clock in MHz

Returns

- NVML_SUCCESS if clockMHz has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or clockMHz is NULL or clockType is invalid
- NVML_ERROR_NOT_SUPPORTED if the device or the clockType on this device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the customer defined maximum boost clock speed specified by the given clock type.

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetSupportedMemoryClocks (nvmlDevice_t device, unsigned int *count, unsigned int *clocksMHz)

Parameters

device
The identifier of the target device

count
Reference in which to provide the clocksMHz array size, and to return the number of elements

clocksMHz
Reference in which to return the clock in MHz

Returns

- NVML_SUCCESS if count and clocksMHz have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or count is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_INSUFFICIENT_SIZE if count is too small (count is set to the number of required elements)
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the list of possible memory clocks that can be used as an argument for nvmlDeviceSetApplicationsClocks.

For Kepler or newer fully supported devices.

See also:

nvmlDeviceSetApplicationsClocks

nvmlDeviceGetSupportedGraphicsClocks

nvmlReturn_t nvmlDeviceGetSupportedGraphicsClocks (nvmlDevice_t device, unsigned int memoryClockMHz, unsigned int *count, unsigned int *clocksMHz)

Parameters

device
The identifier of the target device

memoryClockMHz
Memory clock for which to return possible graphics clocks

count
Reference in which to provide the clocksMHz array size, and to return the number of elements

clocksMHz
Reference in which to return the clocks in MHz

Returns

- NVML_SUCCESS if count and clocksMHz have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_FOUND if the specified memoryClockMHz is not a supported frequency
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or clock is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_INSUFFICIENT_SIZE if count is too small
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the list of possible graphics clocks that can be used as an argument for nvmlDeviceSetApplicationsClocks.

For Kepler or newer fully supported devices.

See also:

nvmlDeviceSetApplicationsClocks

nvmlDeviceGetSupportedMemoryClocks

nvmlReturn_t nvmlDeviceGetAutoBoostedClocksEnabled (nvmlDevice_t device, nvmlEnableState_t *isEnabled, nvmlEnableState_t *defaultIsEnabled)

Parameters

device
The identifier of the target device

isEnabled
Where to store the current state of Auto Boosted clocks of the target device

defaultIsEnabled
Where to store the default Auto Boosted clocks behavior of the target device that the device will revert to when no applications are using the GPU

Returns

- NVML_SUCCESS If isEnabled has been been set with the Auto Boosted clocks state of device
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or isEnabled is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support Auto Boosted clocks
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the current state of Auto Boosted clocks on a device and store it in isEnabled

For Kepler or newer fully supported devices.

Auto Boosted clocks are enabled by default on some hardware, allowing the GPU to run at higher clock rates to maximize performance as thermal limits allow.

On Pascal and newer hardware, Auto Aoosted clocks are controlled through application clocks. Use nvmlDeviceSetApplicationsClocks and nvmlDeviceResetApplicationsClocks to control Auto Boost behavior.

nvmlReturn_t nvmlDeviceGetFanSpeed (nvmlDevice_t device, unsigned int *speed)

Parameters

device
The identifier of the target device

speed
Reference in which to return the fan speed percentage

Returns

- NVML_SUCCESS if speed has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or speed is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not have a fan
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the intended operating speed of the device's fan.

Note: The reported speed is the intended fan speed. If the fan is physically blocked and unable to spin, the output will not match the actual fan speed.

For all discrete products with dedicated fans.

The fan speed is expressed as a percentage of the product's maximum noise tolerance fan speed. This value may exceed 100% in certain cases.

nvmlReturn_t nvmlDeviceGetFanSpeed_v2 (nvmlDevice_t device, unsigned int fan, unsigned int *speed)

Parameters

device
The identifier of the target device

fan
The index of the target fan, zero indexed.

speed
Reference in which to return the fan speed percentage

Returns

- NVML_SUCCESS if speed has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, fan is not an acceptable index, or speed is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not have a fan or is newer than Maxwell
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the intended operating speed of the device's specified fan.

Note: The reported speed is the intended fan speed. If the fan is physically blocked and unable to spin, the output will not match the actual fan speed.

For all discrete products with dedicated fans.

The fan speed is expressed as a percentage of the product's maximum noise tolerance fan speed. This value may exceed 100% in certain cases.

nvmlReturn_t nvmlDeviceGetTargetFanSpeed (nvmlDevice_t device, unsigned int fan, unsigned int *targetSpeed)

Parameters

device
The identifier of the target device
fan
The index of the target fan, zero indexed.
targetSpeed
Reference in which to return the fan speed percentage

Returns

- NVML_SUCCESS if speed has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if device is invalid, fan is not an acceptable index, or speed is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not have a fan or is newer than Maxwell
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the intended target speed of the device's specified fan.

Normally, the driver dynamically adjusts the fan based on the needs of the GPU. But when user set fan speed using nvmlDeviceSetFanSpeed_v2, the driver will attempt to make the fan achieve the setting in nvmlDeviceSetFanSpeed_v2. The actual current speed of the fan is reported in nvmlDeviceGetFanSpeed_v2.

For all discrete products with dedicated fans.

The fan speed is expressed as a percentage of the product's maximum noise tolerance fan speed. This value may exceed 100% in certain cases.

nvmlReturn_t nvmlDeviceGetMinMaxFanSpeed (nvmlDevice_t device, unsigned int *minSpeed, unsigned int *maxSpeed)

Parameters

device
The identifier of the target device
minSpeed
The minimum speed allowed to set
maxSpeed
The maximum speed allowed to set

Description

Retrieves the min and max fan speed that user can set for the GPU fan.

For all cuda-capable discrete products with fans

return NVML_SUCCESS if speed has been adjusted
NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized NVML_ERROR_INVALID_ARGUMENT if device is invalid
NVML_ERROR_NOT_SUPPORTED if the device does not support this (doesn't have fans) NVML_ERROR_UNKNOWN on any unexpected error

nvmlReturn_t nvmlDeviceGetFanControlPolicy_v2 (nvmlDevice_t device, unsigned int fan, nvmlFanControlPolicy_t *policy)

Description

Gets current fan control policy.

For Maxwell or newer fully supported devices.

For all cuda-capable discrete products with fans

device The identifier of the target device policy Reference in which to return the fan control policy

return NVML_SUCCESS if policy has been populated NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized NVML_ERROR_INVALID_ARGUMENT if device is invalid or policy is null or the fan given doesn't reference a fan that exists. NVML_ERROR_NOT_SUPPORTED if the device is older than Maxwell NVML_ERROR_UNKNOWN on any unexpected error

nvmlReturn_t nvmlDeviceGetNumFans (nvmlDevice_t device, unsigned int *numFans)

Parameters

device
The identifier of the target device

numFans
The number of fans

Returns

- NVML_SUCCESS if fan number query was successful
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or numFans is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not have a fan
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the number of fans on the device.

For all discrete products with dedicated fans.

nvmlReturn_t nvmlDeviceGetTemperature (nvmlDevice_t device, nvmlTemperatureSensors_t sensorType, unsigned int *temp)

Parameters

device
The identifier of the target device

sensorType
Flag that indicates which sensor reading to retrieve

temp
Reference in which to return the temperature reading

Returns

- NVML_SUCCESS if temp has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, sensorType is invalid or temp is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not have the specified sensor
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current temperature readings for the device, in degrees C.

For all products.

See nvmlTemperatureSensors_t for details on available temperature sensors.

nvmlReturn_t nvmlDeviceGetTemperatureThreshold (nvmlDevice_t device, nvmlTemperatureThresholds_t thresholdType, unsigned int *temp)

Parameters

device
The identifier of the target device

thresholdType
The type of threshold value queried
temp
Reference in which to return the temperature reading

Returns

- NVML_SUCCESS if temp has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, thresholdType is invalid or temp is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not have a temperature sensor or is unsupported
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the temperature threshold for the GPU with the specified threshold type in degrees C.

For Kepler or newer fully supported devices.

See nvmlTemperatureThresholds_t for details on available temperature thresholds.

Note: This API is no longer the preferred interface for retrieving the following temperature thresholds on Ada and later architectures: NVML_TEMPERATURE_THRESHOLD_SHUTDOWN, NVML_TEMPERATURE_THRESHOLD_SLOWDOWN, NVML_TEMPERATURE_THRESHOLD_MEM_MAX and NVML_TEMPERATURE_THRESHOLD_GPU_MAX.

Support for reading these temperature thresholds for Ada and later architectures would be removed from this API in future releases. Please use nvmlDeviceGetFieldValues with NVML_FI_DEV_TEMPERATURE_* fields to retrieve temperature thresholds on these architectures.

nvmlReturn_t nvmlDeviceGetThermalSettings (nvmlDevice_t device, unsigned int sensorIndex, nvmlGpuThermalSettings_t *pThermalSettings)

Parameters

device
The identifier of the target device
sensorIndex
The index of the thermal sensor
pThermalSettings
Reference in which to return the thermal sensor information

Returns

- NVML_SUCCESS if pThermalSettings has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or pThermalSettings is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Used to execute a list of thermal system instructions.

nvmlReturn_t nvmlDeviceGetPerformanceState (nvmlDevice_t device, nvmlPstates_t *pState)

Parameters

device
The identifier of the target device
pState
Reference in which to return the performance state reading

Returns

- NVML_SUCCESS if pState has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if device is invalid or pState is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current performance state for the device.

For Fermi or newer fully supported devices.

See nvmlPstates_t for details on allowed performance states.

nvmlReturn_t nvmlDeviceGetCurrentClocksEventReasons (nvmlDevice_t device, unsigned long long *clocksEventReasons)

Parameters

device
The identifier of the target device
clocksEventReasons
Reference in which to return bitmask of active clocks event reasons

Returns

- NVML_SUCCESS if clocksEventReasons has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or clocksEventReasons is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves current clocks event reasons.

For all fully supported products.

More than one bit can be enabled at the same time. Multiple reasons can be affecting clocks at once.

See also:

NvmlClocksEventReasons

nvmlDeviceGetSupportedClocksEventReasons

nvmlReturn_t nvmlDeviceGetCurrentClocksThrottleReasons (nvmlDevice_t device, unsigned long long *clocksThrottleReasons)

Description

Deprecated Use nvmlDeviceGetCurrentClocksEventReasons instead

nvmlReturn_t nvmlDeviceGetSupportedClocksEventReasons (nvmlDevice_t device, unsigned long long *supportedClocksEventReasons)

Parameters

device
The identifier of the target device

supportedClocksEventReasons
Reference in which to return bitmask of supported clocks event reasons

Returns

- NVML_SUCCESS if supportedClocksEventReasons has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or supportedClocksEventReasons is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves bitmask of supported clocks event reasons that can be returned by nvmlDeviceGetCurrentClocksEventReasons

For all fully supported products.

This method is not supported in virtual machines running virtual GPU (vGPU).

See also:

NvmlClocksEventReasons

nvmlDeviceGetCurrentClocksEventReasons

nvmlReturn_t nvmlDeviceGetSupportedClocksThrottleReasons (nvmlDevice_t device, unsigned long long *supportedClocksThrottleReasons)

Description

Deprecated Use nvmlDeviceGetSupportedClocksEventReasons instead

nvmlReturn_t nvmlDeviceGetPowerState (nvmlDevice_t device, nvmlPstates_t *pState)

Parameters

device
The identifier of the target device

pState
Reference in which to return the performance state reading

Returns

- NVML_SUCCESS if pState has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or pState is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Deprecated: Use nvmlDeviceGetPerformanceState. This function exposes an incorrect generalization.

Retrieve the current performance state for the device.

For Fermi or newer fully supported devices.

See nvmlPstates_t for details on allowed performance states.

nvmlReturn_t nvmlDeviceGetDynamicPstatesInfo (nvmlDevice_t device, nvmlGpuDynamicPstatesInfo_t *pDynamicPstatesInfo)

Parameters

device
pDynamicPstatesInfo

Returns

- NVML_SUCCESS if pDynamicPstatesInfo has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or pDynamicPstatesInfo is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve performance monitor samples from the associated subdevice.

nvmlReturn_t nvmlDeviceGetMemClkVfOffset (nvmlDevice_t device, int *offset)

Parameters

device
The identifier of the target device
offset
The retrieved MemClk VF offset value

Returns

- NVML_SUCCESS if offset has been successfully queried
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or offset is NULL

- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the MemClk (Memory Clock) VF offset value.

nvmlReturn_t nvmlDeviceGetMinMaxClockOfPState (nvmlDevice_t device, nvmlClockType_t type, nvmlPstates_t pstate, unsigned int *minClockMHz, unsigned int *maxClockMHz)

Parameters

device
The identifier of the target device
type
Clock domain
pstate
PState to query
minClockMHz
Reference in which to return min clock frequency
maxClockMHz
Reference in which to return max clock frequency

Returns

- NVML_SUCCESS if everything worked
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, type or pstate are invalid or both minClockMHz and maxClockMHz are NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature

Description

Retrieve min and max clocks of some clock domain for a given PState

nvmlReturn_t nvmlDeviceGetSupportedPerformanceStates

(nvmlDevice_t device, nvmlPstates_t *pstates, unsigned int size)

Parameters

device
The identifier of the target device
pstates
Container to return the list of performance states supported by device
size
Size of the supplied pstates array in bytes

Returns

- NVML_SUCCESS if pstates array has been retrieved
- NVML_ERROR_INSUFFICIENT_SIZE if the the container supplied was not large enough to hold the resulting list
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device or pstates is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support performance state readings
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get all supported Performance States (P-States) for the device.

The returned array would contain a contiguous list of valid P-States supported by the device. If the number of supported P-States is fewer than the size of the array supplied missing elements would contain NVML_PSTATE_UNKNOWN.

The number of elements in the returned list will never exceed NVML_MAX_GPU_PERF_PSTATES.

nvmlReturn_t nvmlDeviceGetGpcClkMinMaxVfOffset (nvmlDevice_t device, int *minOffset, int *maxOffset)

Parameters

device
The identifier of the target device
minOffset
The retrieved GPCCLK VF min offset value

maxOffset
The retrieved GPCCLK VF max offset value

Returns

- NVML_SUCCESS if offset has been successfully queried
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or offset is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the GPCCLK min max VF offset value.

nvmlReturn_t nvmlDeviceGetMemClkMinMaxVfOffset (nvmlDevice_t device, int *minOffset, int *maxOffset)

Parameters

device
The identifier of the target device
minOffset
The retrieved MemClk VF min offset value
maxOffset
The retrieved MemClk VF max offset value

Returns

- NVML_SUCCESS if offset has been successfully queried
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or offset is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the MemClk (Memory Clock) min max VF offset value.

nvmlReturn_t nvmlDeviceGetPowerManagementMode (nvmlDevice_t device, nvmlEnableState_t *mode)

Parameters

device
The identifier of the target device

mode
Reference in which to return the current power management mode

Returns

- NVML_SUCCESS if mode has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or mode is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

This API has been deprecated.

Retrieves the power management mode associated with this device.

For products from the Fermi family.

- Requires NVML_INFOROM_POWER version 3.0 or higher.

For from the Kepler or newer families.

- Does not require NVML_INFOROM_POWER object.

This flag indicates whether any power management algorithm is currently active on the device. An enabled state does not necessarily mean the device is being actively throttled -- only that that the driver will do so if the appropriate conditions are met.

See nvmlEnableState_t for details on allowed modes.

nvmlReturn_t nvmlDeviceGetPowerManagementLimit (nvmlDevice_t device, unsigned int *limit)

Parameters

device
The identifier of the target device

limit
Reference in which to return the power management limit in milliwatts

Returns

- NVML_SUCCESS if limit has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or limit is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the power management limit associated with this device.

For Fermi or newer fully supported devices.

The power limit defines the upper boundary for the card's power draw. If the card's total power draw reaches this limit the power management algorithm kicks in.

This reading is only available if power management mode is supported. See nvmlDeviceGetPowerManagementMode.

nvmlReturn_t nvmlDeviceGetPowerManagementLimitConstraints (nvmlDevice_t device, unsigned int *minLimit, unsigned int *maxLimit)

Parameters

device
The identifier of the target device

minLimit
Reference in which to return the minimum power management limit in milliwatts

maxLimit
Reference in which to return the maximum power management limit in milliwatts

Returns

- NVML_SUCCESS if minLimit and maxLimit have been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or minLimit or maxLimit is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves information about possible values of power management limits on this device.

For Kepler or newer fully supported devices.

See also:

nvmlDeviceSetPowerManagementLimit

nvmlReturn_t nvmlDeviceGetPowerManagementDefaultLimit (nvmlDevice_t device, unsigned int *defaultLimit)

Parameters

device
The identifier of the target device
defaultLimit
Reference in which to return the default power management limit in milliwatts

Returns

- NVML_SUCCESS if defaultLimit has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or defaultLimit is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible

- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves default power management limit on this device, in milliwatts. Default power management limit is a power management limit that the device boots with.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetPowerUsage (nvmlDevice_t device, unsigned int *power)

Parameters

device
The identifier of the target device

power
Reference in which to return the power usage information

Returns

- NVML_SUCCESS if power has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or power is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support power readings
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves power usage for this GPU in milliwatts and its associated circuitry (e.g. memory)

For Fermi or newer fully supported devices.

On Fermi and Kepler GPUs the reading is accurate to within +/- 5% of current power draw. On Ampere (except GA100) or newer GPUs, the API returns power averaged over 1 sec interval. On GA100 and older architectures, instantaneous power is returned.

See NVML_FI_DEV_POWER_AVERAGE and NVML_FI_DEV_POWER_INSTANT to query specific power values.

It is only available if power management mode is supported. See nvmlDeviceGetPowerManagementMode.

nvmlReturn_t nvmlDeviceGetTotalEnergyConsumption (nvmlDevice_t device, unsigned long long *energy)

Parameters

device
The identifier of the target device

energy
Reference in which to return the energy consumption information

Returns

- NVML_SUCCESS if energy has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or energy is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support energy readings
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves total energy consumption for this GPU in millijoules (mJ) since the driver was last reloaded

For Volta or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetEnforcedPowerLimit (nvmlDevice_t device, unsigned int *limit)

Parameters

device
The device to communicate with

limit
Reference in which to return the power management limit in milliwatts

Returns

- NVML_SUCCESS if limit has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or limit is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature

- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get the effective power limit that the driver enforces after taking into account all limiters

Note: This can be different from the nvmlDeviceGetPowerManagementLimit if other limits are set elsewhere This includes the out of band power limit interface

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetGpuOperationMode (nvmlDevice_t device, nvmlGpuOperationMode_t *current, nvmlGpuOperationMode_t *pending)

Parameters

device
The identifier of the target device

current
Reference in which to return the current GOM

pending
Reference in which to return the pending GOM

Returns

- NVML_SUCCESS if mode has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or current or pending is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current GOM and pending GOM (the one that GPU will switch to after reboot).

For GK110 M-class and X-class Tesla products from the Kepler family. Modes NVML_GOM_LOW_DP and NVML_GOM_ALL_ON are supported on fully supported GeForce products. Not supported on Quadro and Tesla C-class products.

See also:

nvmlGpuOperationMode_t

nvmlDeviceSetGpuOperationMode

nvmlReturn_t nvmlDeviceGetMemoryInfo (nvmlDevice_t device, nvmlMemory_t *memory)

Parameters

device
The identifier of the target device

memory
Reference in which to return the memory information

Returns

- NVML_SUCCESS if memory has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or memory is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the amount of used, free, reserved and total memory available on the device, in bytes. The reserved amount is supported on version 2 only.

For all products.

Enabling ECC reduces the amount of total available memory, due to the extra required parity bits. Under WDDM most device memory is allocated and managed on startup by Windows.

Under Linux and Windows TCC, the reported amount of used memory is equal to the sum of memory allocated by all active channels on the device.

See nvmlMemory_v2_t for details on available memory info.

- In MIG mode, if device handle is provided, the API returns aggregate information, only if the caller has appropriate privileges. Per-instance information can be queried by using specific MIG device handles.

- nvmlDeviceGetMemoryInfo_v2 adds additional memory information.
- On systems where GPUs are NUMA nodes, the accuracy of FB memory utilization provided by this API depends on the memory accounting of the operating system. This is because FB memory is managed by the operating system instead of the NVIDIA GPU driver. Typically, pages allocated from FB memory are not released even after the process terminates to enhance performance. In scenarios where the operating system is under memory pressure, it may resort to utilizing FB memory. Such actions can result in discrepancies in the accuracy of memory reporting.

nvmlReturn_t nvmlDeviceGetComputeMode (nvmlDevice_t device, nvmlComputeMode_t *mode)

Parameters

device
The identifier of the target device

mode
Reference in which to return the current compute mode

Returns

- NVML_SUCCESS if mode has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or mode is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current compute mode for the device.

For all products.

See nvmlComputeMode_t for details on allowed compute modes.

See also:

nvmlDeviceSetComputeMode()

nvmlReturn_t nvmlDeviceGetCudaComputeCapability (nvmlDevice_t device, int *major, int *minor)

Parameters

device
The identifier of the target device

major
Reference in which to return the major CUDA compute capability

minor
Reference in which to return the minor CUDA compute capability

Returns

- NVML_SUCCESS if major and minor have been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or major or minor are NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the CUDA compute capability of the device.

For all products.

Returns the major and minor compute capability version numbers of the device. The major and minor versions are equivalent to the CU_DEVICE_ATTRIBUTE_COMPUTE_CAPABILITY_MINOR and CU_DEVICE_ATTRIBUTE_COMPUTE_CAPABILITY_MAJOR attributes that would be returned by CUDA's cuDeviceGetAttribute().

nvmlReturn_t nvmlDeviceGetEccMode (nvmlDevice_t device, nvmlEnableState_t *current, nvmlEnableState_t *pending)

Parameters

device
The identifier of the target device

current
Reference in which to return the current ECC mode
pending
Reference in which to return the pending ECC mode

Returns

- NVML_SUCCESS if current and pending have been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or either current or pending is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current and pending ECC modes for the device.

For Fermi or newer fully supported devices. Only applicable to devices with ECC. Requires NVML_INFOROM_ECC version 1.0 or higher.

Changing ECC modes requires a reboot. The "pending" ECC mode refers to the target mode following the next reboot.

See nvmlEnableState_t for details on allowed modes.

See also:

nvmlDeviceSetEccMode()

nvmlReturn_t nvmlDeviceGetDefaultEccMode (nvmlDevice_t device, nvmlEnableState_t *defaultMode)

Parameters

device
The identifier of the target device
defaultMode
Reference in which to return the default ECC mode

Returns

- NVML_SUCCESS if current and pending have been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if device is invalid or default is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the default ECC modes for the device.

For Fermi or newer fully supported devices. Only applicable to devices with ECC. Requires NVML_INFOROM_ECC version 1.0 or higher.

See nvmlEnableState_t for details on allowed modes.

See also:

nvmlDeviceSetEccMode()

nvmlReturn_t nvmlDeviceGetBoardId (nvmlDevice_t device, unsigned int *boardId)

Parameters

device
The identifier of the target device

boardId
Reference in which to return the device's board ID

Returns

- NVML_SUCCESS if boardId has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or boardId is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the device boardId from 0-N. Devices with the same boardId indicate GPUs connected to the same PLX. Use in conjunction with nvmlDeviceGetMultiGpuBoard() to decide if they are on the same board as well. The boardId returned is a unique ID for the current configuration. Uniqueness and ordering across reboots and system

configurations is not guaranteed (i.e. if a Tesla K40c returns 0x100 and the two GPUs on a Tesla K10 in the same system returns 0x200 it is not guaranteed they will always return those values but they will always be different from each other).

For Fermi or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetMultiGpuBoard (nvmlDevice_t device, unsigned int *multiGpuBool)

Parameters

device
The identifier of the target device

multiGpuBool
Reference in which to return a zero or non-zero value to indicate whether the device is on a multi GPU board

Returns

- NVML_SUCCESS if multiGpuBool has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or multiGpuBool is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves whether the device is on a Multi-GPU Board Devices that are on multi-GPU boards will set multiGpuBool to a non-zero value.

For Fermi or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetTotalEccErrors (nvmlDevice_t device, nvmlMemoryErrorType_t

errorType, nvmlEccCounterType_t counterType, unsigned long long *eccCounts)

Parameters

device
The identifier of the target device
errorType
Flag that specifies the type of the errors.
counterType
Flag that specifies the counter-type of the errors.
eccCounts
Reference in which to return the specified ECC errors

Returns

- NVML_SUCCESS if eccCounts has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, errorType or counterType is invalid, or eccCounts is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the total ECC error counts for the device.

For Fermi or newer fully supported devices. Only applicable to devices with ECC. Requires NVML_INFOROM_ECC version 1.0 or higher. Requires ECC Mode to be enabled.

The total error count is the sum of errors across each of the separate memory systems, i.e. the total set of errors across the entire device.

See nvmlMemoryErrorType_t for a description of available error types. See nvmlEccCounterType_t for a description of available counter types.

See also:

nvmlDeviceClearEccErrorCounts()

nvmlReturn_t nvmlDeviceGetDetailedEccErrors (nvmlDevice_t device, nvmlMemoryErrorType_t

errorType, nvmlEccCounterType_t counterType, nvmlEccErrorCounts_t *eccCounts)

Parameters

device
The identifier of the target device
errorType
Flag that specifies the type of the errors.
counterType
Flag that specifies the counter-type of the errors.
eccCounts
Reference in which to return the specified ECC errors

Returns

- NVML_SUCCESS if eccCounts has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, errorType or counterType is invalid, or eccCounts is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the detailed ECC error counts for the device.

Deprecated This API supports only a fixed set of ECC error locations On different GPU architectures different locations are supported See nvmlDeviceGetMemoryErrorCounter

For Fermi or newer fully supported devices. Only applicable to devices with ECC. Requires NVML_INFOROM_ECC version 2.0 or higher to report aggregate location-based ECC counts. Requires NVML_INFOROM_ECC version 1.0 or higher to report all other ECC counts. Requires ECC Mode to be enabled.

Detailed errors provide separate ECC counts for specific parts of the memory system.

Reports zero for unsupported ECC error counters when a subset of ECC error counters are supported.

See nvmlMemoryErrorType_t for a description of available bit types. See nvmlEccCounterType_t for a description of available counter types. See nvmlEccErrorCounts_t for a description of provided detailed ECC counts.

See also:

nvmlDeviceClearEccErrorCounts()

nvmlReturn_t nvmlDeviceGetMemoryErrorCounter (nvmlDevice_t device, nvmlMemoryErrorType_t errorType, nvmlEccCounterType_t counterType, nvmlMemoryLocation_t locationType, unsigned long long *count)

Parameters

device
The identifier of the target device
errorType
Flag that specifies the type of error.
counterType
Flag that specifies the counter-type of the errors.
locationType
Specifies the location of the counter.
count
Reference in which to return the ECC counter

Returns

- NVML_SUCCESS if count has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, bitTyp,e counterType or locationType is invalid, or count is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support ECC error reporting in the specified memory
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the requested memory error counter for the device.

For Fermi or newer fully supported devices. Requires NVML_INFOROM_ECC version 2.0 or higher to report aggregate location-based memory error counts. Requires NVML_INFOROM_ECC version 1.0 or higher to report all other memory error counts.

Only applicable to devices with ECC.

Requires ECC Mode to be enabled.

On MIG-enabled GPUs, per instance information can be queried using specific MIG device handles. Per instance information is currently only supported for non-DRAM uncorrectable volatile errors. Querying volatile errors using device handles is currently not supported.

See nvmlMemoryErrorType_t for a description of available memory error types. See nvmlEccCounterType_t for a description of available counter types. See nvmlMemoryLocation_t for a description of available counter locations.

nvmlReturn_t nvmlDeviceGetUtilizationRates (nvmlDevice_t device, nvmlUtilization_t *utilization)

Parameters

device
The identifier of the target device

utilization
Reference in which to return the utilization information

Returns

- NVML_SUCCESS if utilization has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or utilization is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current utilization rates for the device's major subsystems.

For Fermi or newer fully supported devices.

See nvmlUtilization_t for details on available utilization rates.

- During driver initialization when ECC is enabled one can see high GPU and Memory Utilization readings. This is caused by ECC Memory Scrubbing mechanism that is performed during driver initialization.
- On MIG-enabled GPUs, querying device utilization rates is not currently supported.

nvmlReturn_t nvmlDeviceGetEncoderUtilization (nvmlDevice_t device, unsigned int *utilization, unsigned int *samplingPeriodUs)

Parameters

device
The identifier of the target device
utilization
Reference to an unsigned int for encoder utilization info
samplingPeriodUs
Reference to an unsigned int for the sampling period in US

Returns

- NVML_SUCCESS if utilization has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, utilization is NULL, or samplingPeriodUs is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current utilization and sampling size in microseconds for the Encoder

For Kepler or newer fully supported devices.

On MIG-enabled GPUs, querying encoder utilization is not currently supported.

nvmlReturn_t nvmlDeviceGetEncoderCapacity (nvmlDevice_t device, nvmlEncoderType_t encoderQueryType, unsigned int *encoderCapacity)

Parameters

device
The identifier of the target device

encoderQueryType
Type of encoder to query
encoderCapacity
Reference to an unsigned int for the encoder capacity

Returns

- NVML_SUCCESS if encoderCapacity is fetched
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if encoderCapacity is NULL, or device or encoderQueryType are invalid
- NVML_ERROR_NOT_SUPPORTED if device does not support the encoder specified in encodeQueryType
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current capacity of the device's encoder, as a percentage of maximum encoder capacity with valid values in the range 0-100.

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetEncoderStats (nvmlDevice_t device, unsigned int *sessionCount, unsigned int *averageFps, unsigned int *averageLatency)

Parameters

device
The identifier of the target device
sessionCount
Reference to an unsigned int for count of active encoder sessions
averageFps
Reference to an unsigned int for trailing average FPS of all active sessions
averageLatency
Reference to an unsigned int for encode latency in microseconds

Returns

- NVML_SUCCESS if sessionCount, averageFps and averageLatency is fetched
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if sessionCount, or device or averageFps, or averageLatency is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current encoder statistics for a given device.

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetEncoderSessions (nvmlDevice_t device, unsigned int *sessionCount, nvmlEncoderSessionInfo_t *sessionInfos)

Parameters

device
The identifier of the target device
sessionCount
Reference to caller supplied array size, and returns the number of sessions.
sessionInfos
Reference in which to return the session information

Returns

- NVML_SUCCESS if sessionInfos is fetched
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INSUFFICIENT_SIZE if sessionCount is too small, array element count is returned in sessionCount
- NVML_ERROR_INVALID_ARGUMENT if sessionCount is NULL.
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves information about active encoder sessions on a target device.

An array of active encoder sessions is returned in the caller-supplied buffer pointed at by sessionInfos. The array element count is passed in sessionCount, and sessionCount is used to return the number of sessions written to the buffer.

If the supplied buffer is not large enough to accommodate the active session array, the function returns NVML_ERROR_INSUFFICIENT_SIZE, with the element count of nvmlEncoderSessionInfo_t array required in sessionCount. To query the number of active encoder sessions, call this function with *sessionCount = 0. The code will return NVML_SUCCESS with number of active encoder sessions updated in *sessionCount.

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetDecoderUtilization (nvmlDevice_t device, unsigned int *utilization, unsigned int *samplingPeriodUs)

Parameters

device
The identifier of the target device
utilization
Reference to an unsigned int for decoder utilization info
samplingPeriodUs
Reference to an unsigned int for the sampling period in US

Returns

- NVML_SUCCESS if utilization has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, utilization is NULL, or samplingPeriodUs is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current utilization and sampling size in microseconds for the Decoder

For Kepler or newer fully supported devices.

On MIG-enabled GPUs, querying decoder utilization is not currently supported.

nvmlReturn_t nvmlDeviceGetJpgUtilization (nvmlDevice_t device, unsigned int *utilization, unsigned int *samplingPeriodUs)

Parameters

device
The identifier of the target device
utilization
Reference to an unsigned int for jpg utilization info
samplingPeriodUs
Reference to an unsigned int for the sampling period in US

Returns

- NVML_SUCCESS if utilization has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, utilization is NULL, or samplingPeriodUs is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current utilization and sampling size in microseconds for the JPG

TURING_OR_NEWER%

On MIG-enabled GPUs, querying decoder utilization is not currently supported.

nvmlReturn_t nvmlDeviceGetOfaUtilization (nvmlDevice_t device, unsigned int *utilization, unsigned int *samplingPeriodUs)

Parameters

device
The identifier of the target device

utilization
Reference to an unsigned int for ofa utilization info
samplingPeriodUs
Reference to an unsigned int for the sampling period in US

Returns

- NVML_SUCCESS if utilization has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, utilization is NULL, or samplingPeriodUs is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current utilization and sampling size in microseconds for the OFA (Optical Flow Accelerator)

TURING_OR_NEWER%

On MIG-enabled GPUs, querying decoder utilization is not currently supported.

nvmlReturn_t nvmlDeviceGetFBCStats (nvmlDevice_t device, nvmlFBCStats_t *fbcStats)

Parameters

device
The identifier of the target device
fbcStats
Reference to nvmlFBCStats_t structure containing NvFBC stats

Returns

- NVML_SUCCESS if fbcStats is fetched
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if fbcStats is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the active frame buffer capture sessions statistics for a given device.

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetFBCSessions (nvmlDevice_t device, unsigned int *sessionCount, nvmlFBCSessionInfo_t *sessionInfo)

Parameters

device
The identifier of the target device

sessionCount
Reference to caller supplied array size, and returns the number of sessions.

sessionInfo
Reference in which to return the session information

Returns

- NVML_SUCCESS if sessionInfo is fetched
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INSUFFICIENT_SIZE if sessionCount is too small, array element count is returned in sessionCount
- NVML_ERROR_INVALID_ARGUMENT if sessionCount is NULL.
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves information about active frame buffer capture sessions on a target device.

An array of active FBC sessions is returned in the caller-supplied buffer pointed at by sessionInfo. The array element count is passed in sessionCount, and sessionCount is used to return the number of sessions written to the buffer.

If the supplied buffer is not large enough to accommodate the active session array, the function returns NVML_ERROR_INSUFFICIENT_SIZE, with the element count of nvmlFBCSessionInfo_t array required in sessionCount. To query the number of active FBC sessions, call this function with *sessionCount = 0. The code will return NVML_SUCCESS with number of active FBC sessions updated in *sessionCount.

For Maxwell or newer fully supported devices.

hResolution, vResolution, averageFPS and averageLatency data for a FBC session returned in sessionInfo may be zero if there are no new frames captured since the session started.

nvmlReturn_t nvmlDeviceGetDriverModel (nvmlDevice_t device, nvmlDriverModel_t *current, nvmlDriverModel_t *pending)

Parameters

device
The identifier of the target device

current
Reference in which to return the current driver model

pending
Reference in which to return the pending driver model

Returns

- NVML_SUCCESS if either current and/or pending have been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or both current and pending are NULL
- NVML_ERROR_NOT_SUPPORTED if the platform is not windows
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current and pending driver model for the device.

For Fermi or newer fully supported devices. For windows only.

On Windows platforms the device driver can run in either WDDM or WDM (TCC) mode. If a display is attached to the device it must run in WDDM mode. TCC mode is preferred if a display is not attached.

See nvmlDriverModel_t for details on available driver models.

See also:

nvmlDeviceSetDriverModel()

nvmlReturn_t nvmlDeviceGetVbiosVersion (nvmlDevice_t device, char *version, unsigned int length)

Parameters

device
The identifier of the target device
version
Reference to which to return the VBIOS version
length
The maximum allowed length of the string returned in version

Returns

- NVML_SUCCESS if version has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or version is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get VBIOS version of the device.

For all products.

The VBIOS version may change from time to time. It will not exceed 32 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE.

nvmlReturn_t nvmlDeviceGetBridgeChipInfo (nvmlDevice_t device, nvmlBridgeChipHierarchy_t *bridgeHierarchy)

Parameters

device
The identifier of the target device
bridgeHierarchy
Reference to the returned bridge chip Hierarchy

Returns

- NVML_SUCCESS if bridge chip exists
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or bridgeInfo is NULL
- NVML_ERROR_NOT_SUPPORTED if bridge chip not supported on the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get Bridge Chip Information for all the bridge chips on the board.

For all fully supported products. Only applicable to multi-GPU products.

nvmlReturn_t nvmlDeviceGetComputeRunningProcesses_v3 (nvmlDevice_t device, unsigned int *infoCount, nvmlProcessInfo_t *infos)

Parameters

device
The device handle or MIG device handle

infoCount
Reference in which to provide the infos array size, and to return the number of returned elements

infos
Reference in which to return the process information

Returns

- NVML_SUCCESS if infoCount and infos have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INSUFFICIENT_SIZE if infoCount indicates that the infos array is too small infoCount will contain minimal amount of space necessary for the call to complete
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, either of infoCount or infos is NULL

- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get information about processes with a compute context on a device

For Fermi or newer fully supported devices.

This function returns information only about compute running processes (e.g. CUDA application which have active context). Any graphics applications (e.g. using OpenGL, DirectX) won't be listed by this function.

To query the current number of running compute processes, call this function with *infoCount = 0. The return code will be NVML_ERROR_INSUFFICIENT_SIZE, or NVML_SUCCESS if none are running. For this call infos is allowed to be NULL.

The usedGpuMemory field returned is all of the memory used by the application.

Keep in mind that information returned by this call is dynamic and the number of elements might change in time. Allocate more space for infos table in case new compute processes are spawned.

In MIG mode, if device handle is provided, the API returns aggregate information, only if the caller has appropriate privileges. Per-instance information can be queried by using specific MIG device handles. Querying per-instance information using MIG device handles is not supported if the device is in vGPU Host virtualization mode.

See also:

nvmlSystemGetProcessName

nvmlReturn_t nvmlDeviceGetGraphicsRunningProcesses_v3 (nvmlDevice_t device, unsigned int *infoCount, nvmlProcessInfo_t *infos)

Parameters

device
The device handle or MIG device handle

infoCount
Reference in which to provide the infos array size, and to return the number of returned elements

infos
Reference in which to return the process information

Returns

- NVML_SUCCESS if infoCount and infos have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INSUFFICIENT_SIZE if infoCount indicates that the infos array is too small infoCount will contain minimal amount of space necessary for the call to complete
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, either of infoCount or infos is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get information about processes with a graphics context on a device

For Kepler or newer fully supported devices.

This function returns information only about graphics based processes (eg. applications using OpenGL, DirectX)

To query the current number of running graphics processes, call this function with *infoCount = 0. The return code will be NVML_ERROR_INSUFFICIENT_SIZE, or NVML_SUCCESS if none are running. For this call infos is allowed to be NULL.

The usedGpuMemory field returned is all of the memory used by the application.

Keep in mind that information returned by this call is dynamic and the number of elements might change in time. Allocate more space for infos table in case new graphics processes are spawned.

In MIG mode, if device handle is provided, the API returns aggregate information, only if the caller has appropriate privileges. Per-instance information can be queried by using specific MIG device handles. Querying per-instance information using MIG device handles is not supported if the device is in vGPU Host virtualization mode.

See also:

nvmlSystemGetProcessName

nvmlReturn_t nvmlDeviceGetMPSComputeRunningProcesses_v3 (nvmlDevice_t device, unsigned int *infoCount, nvmlProcessInfo_t *infos)

Parameters

device
The device handle or MIG device handle

infoCount
Reference in which to provide the infos array size, and to return the number of returned elements

infos
Reference in which to return the process information

Returns

- NVML_SUCCESS if infoCount and infos have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INSUFFICIENT_SIZE if infoCount indicates that the infos array is too small infoCount will contain minimal amount of space necessary for the call to complete
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, either of infoCount or infos is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get information about processes with a MPS compute context on a device

For Volta or newer fully supported devices.

This function returns information only about compute running processes (e.g. CUDA application which have active context) utilizing MPS. Any graphics applications (e.g. using OpenGL, DirectX) won't be listed by this function.

To query the current number of running compute processes, call this function with *infoCount = 0. The return code will be NVML_ERROR_INSUFFICIENT_SIZE, or NVML_SUCCESS if none are running. For this call infos is allowed to be NULL.

The usedGpuMemory field returned is all of the memory used by the application.

Keep in mind that information returned by this call is dynamic and the number of elements might change in time. Allocate more space for infos table in case new compute processes are spawned.

In MIG mode, if device handle is provided, the API returns aggregate information, only if the caller has appropriate privileges. Per-instance information can be queried by using specific MIG device handles. Querying per-instance information using MIG device handles is not supported if the device is in vGPU Host virtualization mode.

See also:

nvmlSystemGetProcessName

nvmlReturn_t nvmlDeviceGetRunningProcessDetailList (nvmlDevice_t device, nvmlProcessDetailList_t *plist)

Parameters

device
The device handle or MIG device handle

plist
Reference in which to process detail list

Returns

- NVML_SUCCESS if plist->numprocArrayEntries and plist->procArray have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INSUFFICIENT_SIZE if plist->numprocArrayEntries indicates that the plist->procArray is too small plist->numprocArrayEntries will contain minimal amount of space necessary for the call to complete
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, plist is NULL, plist->version is invalid, plist->mode is invalid,
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by device

- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get information about running processes on a device for input context

HOPPER_OR_NEWER%

This function returns information only about running processes (e.g. CUDA application which have active context).

To determine the size of the plist->procArray array to allocate, call the function with plist->numProcArrayEntries set to zero and plist->procArray set to NULL. The return code will be either NVML_ERROR_INSUFFICIENT_SIZE (if there are valid processes of type plist->mode to report on, in which case the plist->numProcArrayEntries field will indicate the required number of entries in the array) or NVML_SUCCESS (if no processes of type plist->mode exist).

The usedGpuMemory field returned is all of the memory used by the application. The usedGpuCcProtectedMemory field returned is all of the protected memory used by the application.

Keep in mind that information returned by this call is dynamic and the number of elements might change in time. Allocate more space for plist->procArray table in case new processes are spawned.

In MIG mode, if device handle is provided, the API returns aggregate information, only if the caller has appropriate privileges. Per-instance information can be queried by using specific MIG device handles. Querying per-instance information using MIG device handles is not supported if the device is in vGPU Host virtualization mode. Protected memory usage is currently not available in MIG mode and in windows.

nvmlReturn_t nvmlDeviceOnSameBoard (nvmlDevice_t device1, nvmlDevice_t device2, int *onSameBoard)

Parameters

device1
The first GPU device

device2
The second GPU device

onSameBoard
Reference in which to return the status. Non-zero indicates that the GPUs are on the same board.

Returns

- NVML_SUCCESS if onSameBoard has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if dev1 or dev2 are invalid or onSameBoard is NULL
- NVML_ERROR_NOT_SUPPORTED if this check is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the either GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Check if the GPU devices are on the same physical board.

For all fully supported products.

nvmlReturn_t nvmlDeviceGetAPIRestriction (nvmlDevice_t device, nvmlRestrictedAPI_t apiType, nvmlEnableState_t *isRestricted)

Parameters

device
The identifier of the target device

apiType
Target API type for this operation

isRestricted
Reference in which to return the current restriction NVML_FEATURE_ENABLED indicates that the API is root-only NVML_FEATURE_DISABLED indicates that the API is accessible to all users

Returns

- NVML_SUCCESS if isRestricted has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, apiType incorrect or isRestricted is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device or the device does not support the feature that is being queried (E.G. Enabling/disabling Auto Boosted clocks is not supported by the device)
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the root/admin permissions on the target API. See nvmlRestrictedAPI_t for the list of supported APIs. If an API is restricted only root users can call that API. See nvmlDeviceSetAPIRestriction to change current permissions.

For all fully supported products.

See also:

nvmlRestrictedAPI_t

nvmlReturn_t nvmlDeviceGetSamples (nvmlDevice_t device, nvmlSamplingType_t type, unsigned long long lastSeenTimeStamp, nvmlValueType_t *sampleValType, unsigned int *sampleCount, nvmlSample_t *samples)

Parameters

device
The identifier for the target device
type
Type of sampling event
lastSeenTimeStamp
Return only samples with timestamp greater than lastSeenTimeStamp.
sampleValType
Output parameter to represent the type of sample value as described in nvmlSampleVal_t
sampleCount
Reference to provide the number of elements which can be queried in samples array
samples
Reference in which samples are returned

Returns

- NVML_SUCCESS if samples are successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, samplesCount is NULL or reference to sampleCount is 0 for non null samples
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_NOT_FOUND if sample entries are not found
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Gets recent samples for the GPU.

For Kepler or newer fully supported devices.

Based on type, this method can be used to fetch the power, utilization or clock samples maintained in the buffer by the driver.

Power, Utilization and Clock samples are returned as type "unsigned int" for the union nvmlValue_t.

To get the size of samples that user needs to allocate, the method is invoked with samples set to NULL. The returned samplesCount will provide the number of samples that can be queried. The user needs to allocate the buffer with size as samplesCount * sizeof(nvmlSample_t).

lastSeenTimeStamp represents CPU timestamp in microseconds. Set it to 0 to fetch all the samples maintained by the underlying buffer. Set lastSeenTimeStamp to one of the timeStamps retrieved from the date of the previous query to get more recent samples.

This method fetches the number of entries which can be accommodated in the provided samples array, and the reference samplesCount is updated to indicate how many samples were actually retrieved. The advantage of using this method for samples in contrast to polling via existing methods is to get get higher frequency data at lower polling cost.

On MIG-enabled GPUs, querying the following sample types, NVML_GPU_UTILIZATION_SAMPLES, NVML_MEMORY_UTILIZATION_SAMPLES NVML_ENC_UTILIZATION_SAMPLES and NVML_DEC_UTILIZATION_SAMPLES, is not currently supported.

nvmlReturn_t nvmlDeviceGetBAR1MemoryInfo (nvmlDevice_t device, nvmlBAR1Memory_t *bar1Memory)

Parameters

device
The identifier of the target device

bar1Memory
Reference in which BAR1 memory information is returned.

Returns

- NVML_SUCCESS if BAR1 memory is successfully retrieved

- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, bar1Memory is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Gets Total, Available and Used size of BAR1 memory.

BAR1 is used to map the FB (device memory) so that it can be directly accessed by the CPU or by 3rd party devices (peer-to-peer on the PCIE bus).

In MIG mode, if device handle is provided, the API returns aggregate information, only if the caller has appropriate privileges. Per-instance information can be queried by using specific MIG device handles.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetViolationStatus (nvmlDevice_t device, nvmlPerfPolicyType_t perfPolicyType, nvmlViolationTime_t *violTime)

Parameters

device
The identifier of the target device

perfPolicyType
Represents Performance policy which can trigger GPU throttling

violTime
Reference to which violation time related information is returned

Returns

- NVML_SUCCESS if violation time is successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, perfPolicyType is invalid, or violTime is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible

Description

Gets the duration of time during which the device was throttled (lower than requested clocks) due to power or thermal constraints.

The method is important to users who are tying to understand if their GPUs throttle at any point during their applications. The difference in violation times at two different reference times gives the indication of GPU throttling event.

Violation for thermal capping is not supported at this time.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetIrqNum (nvmlDevice_t device, unsigned int *irqNum)

Parameters

device
The identifier of the target device

irqNum
The interrupt number associated with the specified device

Returns

- NVML_SUCCESS if irq number is successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or irqNum is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible

Description

Gets the device's interrupt number

nvmlReturn_t nvmlDeviceGetNumGpuCores (nvmlDevice_t device, unsigned int *numCores)

Parameters

device
The identifier of the target device

numCores
The number of cores for the specified device

Returns

- NVML_SUCCESS if Gpu core count is successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or numCores is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible

Description

Gets the device's core count

nvmlReturn_t nvmlDeviceGetPowerSource (nvmlDevice_t device, nvmlPowerSource_t *powerSource)

Parameters

device
The identifier of the target device

powerSource
The power source of the device

Returns

- NVML_SUCCESS if the current power source was successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or powerSource is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible

Description

Gets the devices power source

nvmlReturn_t nvmlDeviceGetMemoryBusWidth (nvmlDevice_t device, unsigned int *busWidth)

Parameters

device
The identifier of the target device
busWidth
The devices's memory bus width

Returns

- NVML_SUCCESS if the memory bus width is successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or busWidth is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible

Description

Gets the device's memory bus width

nvmlReturn_t nvmlDeviceGetPcieLinkMaxSpeed (nvmlDevice_t device, unsigned int *maxSpeed)

Parameters

device
The identifier of the target device
maxSpeed
The devices's PCIE Max Link speed in MBPS

Returns

- NVML_SUCCESS if Pcie Max Link Speed is successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or maxSpeed is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible

Description

Gets the device's PCIE Max Link speed in MBPS

nvmlReturn_t nvmlDeviceGetPcieSpeed (nvmlDevice_t device, unsigned int *pcieSpeed)

Parameters

device
The identifier of the target device
pcieSpeed
The devices's PCIe Max Link speed in Mbps

Returns

- NVML_SUCCESS if pcieSpeed has been retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or pcieSpeed is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support PCIe speed getting
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Gets the device's PCIe Link speed in Mbps

nvmlReturn_t nvmlDeviceGetAdaptiveClockInfoStatus (nvmlDevice_t device, unsigned int *adaptiveClockStatus)

Parameters

device
The identifier of the target device
adaptiveClockStatus
The current adaptive clocking status, either NVML_ADAPTIVE_CLOCKING_INFO_STATUS_DISABLED or NVML_ADAPTIVE_CLOCKING_INFO_STATUS_ENABLED

Returns

- NVML_SUCCESS if the current adaptive clocking status is successfully retrieved

- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or adaptiveClockStatus is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible

Description

Gets the device's Adaptive Clock status

nvmlReturn_t nvmlDeviceGetBusType (nvmlDevice_t device, nvmlBusType_t *type)

Parameters

device
The identifier of the target device
type
The PCI Bus type

Description

Get the type of the GPU Bus (PCIe, PCI, ...)

return

- NVML_SUCCESS if the bus type is successfully retreived
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if is invalid or is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

nvmlReturn_t nvmlDeviceGetGpuFabricInfo (nvmlDevice_t device, nvmlGpuFabricInfo_t *gpuFabricInfo)

Parameters

device
The identifier of the target device
gpuFabricInfo
Information about GPU fabric state

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_NOT_SUPPORTED If device doesn't support gpu fabric

Description

Deprecated: Will be deprecated in a future release. Use nvmlDeviceGetGpuFabricInfoV instead

Get fabric information associated with the device.

HOPPER_OR_NEWER%

On Hopper + NVSwitch systems, GPU is registered with the NVIDIA Fabric Manager Upon successful registration, the GPU is added to the NVLink fabric to enable peer-to-peer communication. This API reports the current state of the GPU in the NVLink fabric along with other useful information.

nvmlReturn_t nvmlDeviceGetGpuFabricInfoV (nvmlDevice_t device, nvmlGpuFabricInfoV_t *gpuFabricInfo)

Parameters

device
The identifier of the target device
gpuFabricInfo
Information about GPU fabric state

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_NOT_SUPPORTED If device doesn't support gpu fabric

Description

Versioned wrapper around nvmlDeviceGetGpuFabricInfo that accepts a versioned nvmlGpuFabricInfo_v2_t or later output structure.

The caller must set the nvmlGpuFabricInfoV_t::version field to the appropriate version prior to calling this function. For example:

```
nvmlGpuFabricInfoV_t fabricInfo =
          { .version = nvmlGpuFabricInfo_v2 };
nvmlReturn_t result
 = nvmlDeviceGetGpuFabricInfoV(device,&fabricInfo);
```

HOPPER_OR_NEWER%

nvmlReturn_t nvmlSystemGetConfComputeCapabilities (nvmlConfComputeSystemCaps_t *capabilities)

Parameters

capabilities
System CC capabilities

Returns

- NVML_SUCCESS if capabilities were successfully queried
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if capabilities is invalid
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device

Description

Get Conf Computing System capabilities.

For Ampere or newer fully supported devices. Supported on Linux, Windows TCC.

nvmlReturn_t nvmlSystemGetConfComputeState (nvmlConfComputeSystemState_t *state)

Parameters

state
System CC State

Returns

- NVML_SUCCESS if state were successfully queried
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if state is invalid
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device

Description

Get Conf Computing System State.

For Ampere or newer fully supported devices. Supported on Linux, Windows TCC.

nvmlReturn_t nvmlDeviceGetConfComputeMemSizeInfo (nvmlDevice_t device, nvmlConfComputeMemSizeInfo_t *memInfo)

Parameters

device
Device handle
memInfo
Protected/Unprotected Memory sizes

Returns

- NVML_SUCCESS if memInfo were successfully queried
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if memInfo or device is invalid
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device

Description

Get Conf Computing Protected and Unprotected Memory Sizes.

For Ampere or newer fully supported devices. Supported on Linux, Windows TCC.

nvmlReturn_t nvmlSystemGetConfComputeGpusReadyState (unsigned int *isAcceptingWork)

Parameters

isAcceptingWork
Returns GPU current work accepting state,
NVML_CC_ACCEPTING_CLIENT_REQUESTS_TRUE or
NVML_CC_ACCEPTING_CLIENT_REQUESTS_FALSE

Description

Get Conf Computing GPUs ready state.

For Ampere or newer fully supported devices. Supported on Linux, Windows TCC.

return

- NVML_SUCCESS if current GPUs ready state were successfully queried

- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if isAcceptingWork is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device

nvmlReturn_t nvmlDeviceGetConfComputeProtectedMemoryUsage (nvmlDevice_t device, nvmlMemory_t *memory)

Parameters

device
The identifier of the target device
memory
Reference in which to return the memory information

Returns

- NVML_SUCCESS if memory has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or memory is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get Conf Computing protected memory usage.

For Ampere or newer fully supported devices. Supported on Linux, Windows TCC.

nvmlReturn_t nvmlDeviceGetConfComputeGpuCertificate (nvmlDevice_t device, nvmlConfComputeGpuCertificate_t *gpuCert)

Parameters

device
The identifier of the target device
gpuCert
Reference in which to return the gpu certificate information

Returns

- NVML_SUCCESS if gpu certificate info has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or memory is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get Conf Computing Gpu certificate details.

For Ampere or newer fully supported devices. Supported on Linux, Windows TCC.

nvmlReturn_t nvmlDeviceGetConfComputeGpuAttestationReport (nvmlDevice_t device, nvmlConfComputeGpuAttestationReport_t *gpuAtstReport)

Parameters

device
The identifier of the target device

gpuAtstReport
Reference in which to return the gpu attestation report

Returns

- NVML_SUCCESS if gpu attestation report has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or memory is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get Conf Computing Gpu attestation report.

For Ampere or newer fully supported devices. Supported on Linux, Windows TCC.

nvmlReturn_t nvmlSystemGetConfComputeKeyRotationThresholdInfo

(nvmlConfComputeGetKeyRotationThresholdInfo_t *pKeyRotationThrInfo)

Parameters

pKeyRotationThrInfo
Reference in which to return the key rotation threshold data

Returns

- NVML_SUCCESS if gpu key rotation threshold info has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or memory is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get Conf Computing key rotation threshold detail.

HOPPER_OR_NEWER% Supported on Linux, Windows TCC.

nvmlReturn_t nvmlSystemGetConfComputeSettings (nvmlSystemConfComputeSettings_t *settings)

Parameters

settings
System CC settings

Returns

- NVML_SUCCESS if the query is success
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or counters is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_ARGUMENT_VERSION_MISMATCH if the provided version is invalid/unsupported
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get Conf Computing System Settings.

HOPPER_OR_NEWER% Supported on Linux, Windows TCC.

nvmlReturn_t nvmlDeviceGetGspFirmwareVersion (nvmlDevice_t device, char *version)

Parameters

device
Device handle
version
The retrieved GSP firmware version

Returns

- NVML_SUCCESS if GSP firmware version is sucessfully retrieved
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or GSP version pointer is NULL
- NVML_ERROR_NOT_SUPPORTED if GSP firmware is not enabled for GPU
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve GSP firmware version.

The caller passes in buffer via version and corresponding GSP firmware numbered version is returned with the same parameter in string format.

nvmlReturn_t nvmlDeviceGetGspFirmwareMode (nvmlDevice_t device, unsigned int *isEnabled, unsigned int *defaultMode)

Parameters

device
Device handle
isEnabled
Pointer to specify if GSP firmware is enabled
defaultMode
Pointer to specify if GSP firmware is supported by default on device

Returns

- NVML_SUCCESS if GSP firmware mode is sucessfully retrieved
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or any of isEnabled or defaultMode is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve GSP firmware mode.

The caller passes in integer pointers. GSP firmware enablement and default mode information is returned with corresponding parameters. The return value in isEnabled and defaultMode should be treated as boolean.

nvmlReturn_t nvmlDeviceGetRetiredPages (nvmlDevice_t device, nvmlPageRetirementCause_t cause, unsigned int *pageCount, unsigned long long *addresses)

Parameters

device
The identifier of the target device

cause
Filter page addresses by cause of retirement

pageCount
Reference in which to provide the addresses buffer size, and to return the number of retired pages that match cause Set to 0 to query the size without allocating an addresses buffer

addresses
Buffer to write the page addresses into

Returns

- NVML_SUCCESS if pageCount was populated and addresses was filled
- NVML_ERROR_INSUFFICIENT_SIZE if pageCount indicates the buffer is not large enough to store all the matching page addresses. pageCount is set to the needed size.
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, pageCount is NULL, cause is invalid, or addresses is NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature

- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Returns the list of retired pages by source, including pages that are pending retirement The address information provided from this API is the hardware address of the page that was retired. Note that this does not match the virtual address used in CUDA, but will match the address information in XID 63

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetRetiredPages_v2 (nvmlDevice_t device, nvmlPageRetirementCause_t cause, unsigned int *pageCount, unsigned long long *addresses, unsigned long long *timestamps)

Parameters

device
The identifier of the target device

cause
Filter page addresses by cause of retirement

pageCount
Reference in which to provide the addresses buffer size, and to return the number of retired pages that match cause Set to 0 to query the size without allocating an addresses buffer

addresses
Buffer to write the page addresses into

timestamps
Buffer to write the timestamps of page retirement, additional for _v2

Returns

- NVML_SUCCESS if pageCount was populated and addresses was filled
- NVML_ERROR_INSUFFICIENT_SIZE if pageCount indicates the buffer is not large enough to store all the matching page addresses. pageCount is set to the needed size.
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, pageCount is NULL, cause is invalid, or addresses is NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature

- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Returns the list of retired pages by source, including pages that are pending retirement The address information provided from this API is the hardware address of the page that was retired. Note that this does not match the virtual address used in CUDA, but will match the address information in XID 63

nvmlDeviceGetRetiredPages_v2 adds an additional timestamps parameter to return the time of each page's retirement.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetRetiredPagesPendingStatus (nvmlDevice_t device, nvmlEnableState_t *isPending)

Parameters

device
The identifier of the target device

isPending
Reference in which to return the pending status

Returns

- NVML_SUCCESS if isPending was populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or isPending is NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Check if any pages are pending retirement and need a reboot to fully retire.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetRemappedRows (nvmlDevice_t device, unsigned int *corrRows, unsigned int *uncRows, unsigned int *isPending, unsigned int *failureOccurred)

Parameters

device
The identifier of the target device
corrRows
Reference for number of rows remapped due to correctable errors
uncRows
Reference for number of rows remapped due to uncorrectable errors
isPending
Reference for whether or not remappings are pending
failureOccurred
Reference that is set when a remapping has failed in the past

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_INVALID_ARGUMENT If corrRows, uncRows, isPending or failureOccurred is invalid
- NVML_ERROR_NOT_SUPPORTED If MIG is enabled or if the device doesn't support this feature
- NVML_ERROR_UNKNOWN Unexpected error

Description

Get number of remapped rows. The number of rows reported will be based on the cause of the remapping. isPending indicates whether or not there are pending remappings. A reset will be required to actually remap the row. failureOccurred will be set if a row remapping ever failed in the past. A pending remapping won't affect future work on the GPU since error-containment and dynamic page blacklisting will take care of that.

On MIG-enabled GPUs with active instances, querying the number of remapped rows is not supported

For Ampere or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetRowRemapperHistogram (nvmlDevice_t device, nvmlRowRemapperHistogramValues_t *values)

Parameters

device
Device handle
values
Histogram values

Returns

- NVML_SUCCESS On success
- NVML_ERROR_UNKNOWN On any unexpected error

Description

Get the row remapper histogram. Returns the remap availability for each bank on the GPU.

nvmlReturn_t nvmlDeviceGetArchitecture (nvmlDevice_t device, nvmlDeviceArchitecture_t *arch)

Parameters

device
The identifier of the target device
arch
Reference where architecture is returned, if call successful. Set to NVML_DEVICE_ARCH_* upon success

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device or arch (output refererence) are invalid

Description

Get architecture for device

nvmlReturn_t nvmlDeviceGetClkMonStatus (nvmlDevice_t device, nvmlClkMonStatus_t *status)

Parameters

device
The identifier of the target device

status
Reference in which to return the clkmon fault status

Returns

- NVML_SUCCESS if status has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or status is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the frequency monitor fault status for the device.

For Ampere or newer fully supported devices. Requires root user.

See nvmlClkMonStatus_t for details on decoding the status output.

See also:

nvmlDeviceGetClkMonStatus()

nvmlReturn_t nvmlDeviceGetProcessUtilization (nvmlDevice_t device, nvmlProcessUtilizationSample_t *utilization, unsigned int *processSamplesCount, unsigned long long lastSeenTimeStamp)

Parameters

device
The identifier of the target device

utilization
Pointer to caller-supplied buffer in which guest process utilization samples are returned

processSamplesCount
Pointer to caller-supplied array size, and returns number of processes running

lastSeenTimeStamp
Return only samples with timestamp greater than lastSeenTimeStamp.

Returns

- NVML_SUCCESS if utilization has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, utilization is NULL, or samplingPeriodUs is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_NOT_FOUND if sample entries are not found
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current utilization and process ID

For Maxwell or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, and video decoder for processes running. Utilization values are returned as an array of utilization sample structures in the caller-supplied buffer pointed at by utilization. One utilization sample structure is returned per process running, that had some non-zero utilization during the last sample period. It includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "unsigned int" values. If no valid sample entries are found since the lastSeenTimeStamp, NVML_ERROR_NOT_FOUND is returned.

To read utilization values, first determine the size of buffer required to hold the samples by invoking the function with utilization set to NULL. The caller should allocate a buffer of size processSamplesCount * sizeof(nvmlProcessUtilizationSample_t). Invoke the function again with the allocated buffer passed in utilization, and processSamplesCount set to the number of entries the buffer is sized for.

On successful return, the function updates processSamplesCount with the number of process utilization sample structures that were actually written. This may differ from a previously read value as instances are created or destroyed.

lastSeenTimeStamp represents the CPU timestamp in microseconds at which utilization samples were last read. Set it to 0 to read utilization based on all the samples maintained

by the driver's internal sample buffer. Set lastSeenTimeStamp to a timeStamp retrieved from a previous query to read utilization since the previous query.

On MIG-enabled GPUs, querying process utilization is not currently supported.

nvmlReturn_t nvmlDeviceGetProcessesUtilizationInfo (nvmlDevice_t device, nvmlProcessesUtilizationInfo_t *procesesUtilInfo)

Parameters

device
The identifier of the target device

procesesUtilInfo
Pointer to the caller-provided structure of nvmlProcessesUtilizationInfo_t.

Returns

- NVML_SUCCESS if procesesUtilInfo->procUtilArray has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or procesesUtilInfo is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_NOT_FOUND if sample entries are not found
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_VERSION_MISMATCH if the version of procesesUtilInfo is invalid
- NVML_ERROR_INSUFFICIENT_SIZE if procesesUtilInfo->procUtilArray is NULL, or the buffer size of procesesUtilInfo->procUtilArray is too small. The caller should check the minimul array size from the returned procesesUtilInfo->processSamplesCount, and call the function again with a buffer no smaller than procesesUtilInfo->processSamplesCount * sizeof(nvmlProcessUtilizationInfo_t)
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the recent utilization and process ID for all running processes

For Maxwell or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, and video decoder, jpeg decoder, OFA (Optical Flow Accelerator) for all running processes. Utilization values are returned as an array of utilization sample structures in the caller-

supplied buffer pointed at by procesesUtilInfo->procUtilArray. One utilization sample structure is returned per process running, that had some non-zero utilization during the last sample period. It includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "unsigned int" values.

The caller should allocate a buffer of size processSamplesCount * sizeof(nvmlProcessUtilizationInfo_t). If the buffer is too small, the API will return NVML_ERROR_INSUFFICIENT_SIZE, with the recommended minimal buffer size at procesesUtilInfo->processSamplesCount. The caller should invoke the function again with the allocated buffer passed in procesesUtilInfo->procUtilArray, and procesesUtilInfo->processSamplesCount set to the number no less than the recommended value by the previous API return.

On successful return, the function updates procesesUtilInfo->processSamplesCount with the number of process utilization info structures that were actually written. This may differ from a previously read value as instances are created or destroyed.

procesesUtilInfo->lastSeenTimeStamp represents the CPU timestamp in microseconds at which utilization samples were last read. Set it to 0 to read utilization based on all the samples maintained by the driver's internal sample buffer. Set procesesUtilInfo->lastSeenTimeStamp to a timeStamp retrieved from a previous query to read utilization since the previous query.

procesesUtilInfo->version is the version number of the structure nvmlProcessesUtilizationInfo_t, the caller should set the correct version number to retrieve the specific version of processes utilization information.

On MIG-enabled GPUs, querying process utilization is not currently supported.

4.15.1. CPU and Memory Affinity

Device Queries

This chapter describes NVML operations that are associated with CPU and memory affinity.

nvmlReturn_t nvmlDeviceGetMemoryAffinity (nvmlDevice_t device, unsigned int nodeSetSize, unsignedlong *nodeSet, nvmlAffinityScope_t scope)

Parameters

device

The identifier of the target device

nodeSetSize
The size of the nodeSet array that is safe to access
nodeSet
Array reference in which to return a bitmask of NODEs, 64 NODEs per unsigned long on 64-bit machines, 32 on 32-bit machines
scope
Scope that change the default behavior

Returns

- NVML_SUCCESS if NUMA node Affinity has been filled
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, nodeSetSize == 0, nodeSet is NULL or scope is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves an array of unsigned ints (sized to nodeSetSize) of bitmasks with the ideal memory affinity within node or socket for the device. For example, if NUMA node 0, 1 are ideal within the socket for the device and nodeSetSize == 1, result[0] = 0x3

If requested scope is not applicable to the target topology, the API will fall back to reporting the memory affinity for the immediate non-I/O ancestor of the device.

For Kepler or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetCpuAffinityWithinScope (nvmlDevice_t device, unsigned int cpuSetSize, unsignedlong *cpuSet, nvmlAffinityScope_t scope)

Parameters

device
The identifier of the target device
cpuSetSize
The size of the cpuSet array that is safe to access
cpuSet
Array reference in which to return a bitmask of CPUs, 64 CPUs per unsigned long on 64-bit machines, 32 on 32-bit machines

scope
Scope that change the default behavior

Returns

- NVML_SUCCESS if cpuAffinity has been filled
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, cpuSetSize == 0, cpuSet is NULL or sope is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves an array of unsigned ints (sized to cpuSetSize) of bitmasks with the ideal CPU affinity within node or socket for the device. For example, if processors 0, 1, 32, and 33 are ideal for the device and cpuSetSize == 2, result[0] = 0x3, result[1] = 0x3

If requested scope is not applicable to the target topology, the API will fall back to reporting the CPU affinity for the immediate non-I/O ancestor of the device.

For Kepler or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetCpuAffinity (nvmlDevice_t device, unsigned int cpuSetSize, unsignedlong *cpuSet)

Parameters

device
The identifier of the target device

cpuSetSize
The size of the cpuSet array that is safe to access

cpuSet
Array reference in which to return a bitmask of CPUs, 64 CPUs per unsigned long on 64-bit machines, 32 on 32-bit machines

Returns

- NVML_SUCCESS if cpuAffinity has been filled
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, cpuSetSize == 0, or cpuSet is NULL

- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves an array of unsigned ints (sized to cpuSetSize) of bitmasks with the ideal CPU affinity for the device For example, if processors 0, 1, 32, and 33 are ideal for the device and cpuSetSize == 2, result[0] = 0x3, result[1] = 0x3 This is equivalent to calling nvmlDeviceGetCpuAffinityWithinScope with NVML_AFFINITY_SCOPE_NODE.

For Kepler or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceSetCpuAffinity (nvmlDevice_t device)

Parameters

device
The identifier of the target device

Returns

- NVML_SUCCESS if the calling process has been successfully bound
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Sets the ideal affinity for the calling thread and device using the guidelines given in nvmlDeviceGetCpuAffinity(). Note, this is a change as of version 8.0. Older versions set the affinity for a calling process and all children. Currently supports up to 1024 processors.

For Kepler or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceClearCpuAffinity (nvmlDevice_t device)

Parameters

device
The identifier of the target device

Returns

- NVML_SUCCESS if the calling process has been successfully unbound
- NVML_ERROR_INVALID_ARGUMENT if device is invalid
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Clear all affinity bindings for the calling thread. Note, this is a change as of version 8.0 as older versions cleared the affinity for a calling process and all children.

For Kepler or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetNumaNodeId (nvmlDevice_t device, unsigned int *node)

Parameters

device
The device handle
node
NUMA node ID of the device

Returns

- NVML_SUCCESS if the NUMA node is retrieved successfully
- NVML_ERROR_NOT_SUPPORTED if request is not supported on the current platform
- NVML_ERROR_INVALID_ARGUMENT if device node is invalid

Description

Get the NUMA node of the given GPU device. This only applies to platforms where the GPUs are NUMA nodes.

#define NVML_AFFINITY_SCOPE_NODE 0

Scope of NUMA node for affinity queries.

#define NVML_AFFINITY_SCOPE_SOCKET 1

Scope of processor socket for affinity queries.

4.16. Unit Commands

This chapter describes NVML operations that change the state of the unit. For S-class products. Each of these requires root/admin access. Non-admin users will see an NVML_ERROR_NO_PERMISSION error code when invoking any of these methods.

nvmlReturn_t nvmlUnitSetLedState (nvmlUnit_t unit, nvmlLedColor_t color)

Parameters

unit
The identifier of the target unit

color
The target LED color

Returns

- NVML_SUCCESS if the LED color has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if unit or color is invalid
- NVML_ERROR_NOT_SUPPORTED if this is not an S-class product
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set the LED state for the unit. The LED can be either green (0) or amber (1).

For S-class products. Requires root/admin permissions.

This operation takes effect immediately.

Current S-Class products don't provide unique LEDs for each unit. As such, both front and back LEDs will be toggled in unison regardless of which unit is specified with this command.

See nvmlLedColor_t for available colors.

See also:

nvmlUnitGetLedState()

4.17. Device Commands

This chapter describes NVML operations that change the state of the device.
Each of these requires root/admin access. Non-admin users will see an
NVML_ERROR_NO_PERMISSION error code when invoking any of these methods.

nvmlReturn_t nvmlDeviceSetPersistenceMode (nvmlDevice_t device, nvmlEnableState_t mode)

Parameters

device
The identifier of the target device
mode
The target persistence mode

Returns

- NVML_SUCCESS if the persistence mode was set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or mode is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set the persistence mode for the device.

For all products. For Linux only. Requires root/admin permissions.

The persistence mode determines whether the GPU driver software is torn down after the last client exits.

This operation takes effect immediately. It is not persistent across reboots. After each reboot the persistence mode is reset to "Disabled".

See nvmlEnableState_t for available modes.

After calling this API with mode set to NVML_FEATURE_DISABLED on a device that has its own NUMA memory, the given device handle will no longer be valid, and to continue to interact with this device, a new handle should be obtained from one of the nvmlDeviceGetHandleBy*() APIs. This limitation is currently only applicable to devices that have a coherent NVLink connection to system memory.

See also:

nvmlDeviceGetPersistenceMode()

nvmlReturn_t nvmlDeviceSetComputeMode (nvmlDevice_t device, nvmlComputeMode_t mode)

Parameters

device
The identifier of the target device

mode
The target compute mode

Returns

- NVML_SUCCESS if the compute mode was set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or mode is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set the compute mode for the device.

For all products. Requires root/admin permissions.

The compute mode determines whether a GPU can be used for compute operations and whether it can be shared across contexts.

This operation takes effect immediately. Under Linux it is not persistent across reboots and always resets to "Default". Under windows it is persistent.

Under windows compute mode may only be set to DEFAULT when running in WDDM

On MIG-enabled GPUs, compute mode would be set to DEFAULT and changing it is not supported.

See nvmlComputeMode_t for details on available compute modes.

See also:

nvmlDeviceGetComputeMode()

nvmlReturn_t nvmlDeviceSetEccMode (nvmlDevice_t device, nvmlEnableState_t ecc)

Parameters

device
The identifier of the target device

ecc
The target ECC mode

Returns

- NVML_SUCCESS if the ECC mode was set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or ecc is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set the ECC mode for the device.

For Kepler or newer fully supported devices. Only applicable to devices with ECC. Requires NVML_INFOROM_ECC version 1.0 or higher. Requires root/admin permissions.

The ECC mode determines whether the GPU enables its ECC support.

This operation takes effect after the next reboot.

See nvmlEnableState_t for details on available modes.

See also:

nvmlDeviceGetEccMode()

nvmlReturn_t nvmlDeviceClearEccErrorCounts (nvmlDevice_t device, nvmlEccCounterType_t counterType)

Parameters

device
The identifier of the target device

counterType
Flag that indicates which type of errors should be cleared.

Returns

- NVML_SUCCESS if the error counts were cleared
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or counterType is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Clear the ECC error and other memory error counts for the device.

For Kepler or newer fully supported devices. Only applicable to devices with ECC. Requires NVML_INFOROM_ECC version 2.0 or higher to clear aggregate location-based ECC counts. Requires NVML_INFOROM_ECC version 1.0 or higher to clear all other ECC counts. Requires root/admin permissions. Requires ECC Mode to be enabled.

Sets all of the specified ECC counters to 0, including both detailed and total counts.

This operation takes effect immediately.

See nvmlMemoryErrorType_t for details on available counter types.

See also:

- nvmlDeviceGetDetailedEccErrors()
- nvmlDeviceGetTotalEccErrors()

nvmlReturn_t nvmlDeviceSetDriverModel (nvmlDevice_t device, nvmlDriverModel_t driverModel, unsigned int flags)

Parameters

device
The identifier of the target device
driverModel
The target driver model
flags
Flags that change the default behavior

Returns

- NVML_SUCCESS if the driver model has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or driverModel is invalid
- NVML_ERROR_NOT_SUPPORTED if the platform is not windows or the device does not support this feature
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set the driver model for the device.

For Fermi or newer fully supported devices. For windows only. Requires root/admin permissions.

On Windows platforms the device driver can run in either WDDM or WDM (TCC) mode. If a display is attached to the device it must run in WDDM mode.

It is possible to force the change to WDM (TCC) while the display is still attached with a force flag (nvmlFlagForce). This should only be done if the host is subsequently powered down and the display is detached from the device before the next reboot.

This operation takes effect after the next reboot.

Windows driver model may only be set to WDDM when running in DEFAULT compute mode.

Change driver model to WDDM is not supported when GPU doesn't support graphics acceleration or will not support it after reboot. See nvmlDeviceSetGpuOperationMode.

See nvmlDriverModel_t for details on available driver models. See nvmlFlagDefault and nvmlFlagForce

See also:

nvmlDeviceGetDriverModel()

nvmlReturn_t nvmlDeviceSetGpuLockedClocks (nvmlDevice_t device, unsigned int minGpuClockMHz, unsigned int maxGpuClockMHz)

Parameters

device
The identifier of the target device
minGpuClockMHz
Requested minimum gpu clock in MHz
maxGpuClockMHz
Requested maximum gpu clock in MHz

Returns

- NVML_SUCCESS if new settings were successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or minGpuClockMHz and maxGpuClockMHz is not a valid clock combination
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set clocks that device will lock to.

Sets the clocks that the device will be running at to the value in the range of minGpuClockMHz to maxGpuClockMHz. Setting this will supersede application clock values and take effect regardless if a cuda app is running. See /ref nvmlDeviceSetApplicationsClocks

Can be used as a setting to request constant performance.

This can be called with a pair of integer clock frequencies in MHz, or a pair of /ref nvmlClockLimitId_t values. See the table below for valid combinations of these values.

minGpuClock | maxGpuClock | Effect ----------+---------- +---------- tdp | tdp | Lock clock to TDP unlimited | tdp | Upper bound is TDP but clock may drift below this tdp | unlimited | Lower bound is TDP but clock may boost above this unlimited | unlimited | Unlocked (== nvmlDeviceResetGpuLockedClocks)

If one arg takes one of these values, the other must be one of these values as well. Mixed numeric and symbolic calls return NVML_ERROR_INVALID_ARGUMENT.

Requires root/admin permissions.

After system reboot or driver reload applications clocks go back to their default value. See nvmlDeviceResetGpuLockedClocks.

For Volta or newer fully supported devices.

nvmlReturn_t nvmlDeviceResetGpuLockedClocks (nvmlDevice_t device)

Parameters

device
The identifier of the target device

Returns

- NVML_SUCCESS if new settings were successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Resets the gpu clock to the default value

This is the gpu clock that will be used after system reboot or driver reload. Default values are idle clocks, but the current values can be changed using nvmlDeviceSetApplicationsClocks.

See also:

nvmlDeviceSetGpuLockedClocks

For Volta or newer fully supported devices.

nvmlReturn_t nvmlDeviceSetMemoryLockedClocks (nvmlDevice_t device, unsigned int minMemClockMHz, unsigned int maxMemClockMHz)

Parameters

device
The identifier of the target device
minMemClockMHz
Requested minimum memory clock in MHz
maxMemClockMHz
Requested maximum memory clock in MHz

Returns

- NVML_SUCCESS if new settings were successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or minGpuClockMHz and maxGpuClockMHz is not a valid clock combination
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set memory clocks that device will lock to.

Sets the device's memory clocks to the value in the range of minMemClockMHz to maxMemClockMHz. Setting this will supersede application clock values and take effect regardless of whether a cuda app is running. See /ref nvmlDeviceSetApplicationsClocks

Can be used as a setting to request constant performance.

Requires root/admin permissions.

After system reboot or driver reload applications clocks go back to their default value. See nvmlDeviceResetMemoryLockedClocks.

For Ampere or newer fully supported devices.

nvmlReturn_t nvmlDeviceResetMemoryLockedClocks (nvmlDevice_t device)

Parameters

device
The identifier of the target device

Returns

- NVML_SUCCESS if new settings were successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Resets the memory clock to the default value

This is the memory clock that will be used after system reboot or driver reload. Default values are idle clocks, but the current values can be changed using nvmlDeviceSetApplicationsClocks.

See also:

nvmlDeviceSetMemoryLockedClocks

For Ampere or newer fully supported devices.

nvmlReturn_t nvmlDeviceSetApplicationsClocks (nvmlDevice_t device, unsigned int memClockMHz, unsigned int graphicsClockMHz)

Parameters

device
The identifier of the target device
memClockMHz
Requested memory clock in MHz
graphicsClockMHz
Requested graphics clock in MHz

Returns

- NVML_SUCCESS if new settings were successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or memClockMHz and graphicsClockMHz is not a valid clock combination
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set clocks that applications will lock to.

Sets the clocks that compute and graphics applications will be running at. e.g. CUDA driver requests these clocks during context creation which means this property defines clocks at which CUDA applications will be running unless some overspec event occurs (e.g. over power, over thermal or external HW brake).

Can be used as a setting to request constant performance.

On Pascal and newer hardware, this will automatically disable automatic boosting of clocks.

On K80 and newer Kepler and Maxwell GPUs, users desiring fixed performance should also call nvmlDeviceSetAutoBoostedClocksEnabled to prevent clocks from automatically boosting above the clock value being set.

For Kepler or newer non-GeForce fully supported devices and Maxwell or newer GeForce devices. Requires root/admin permissions.

See nvmlDeviceGetSupportedMemoryClocks and nvmlDeviceGetSupportedGraphicsClocks for details on how to list available clocks combinations.

After system reboot or driver reload applications clocks go back to their default value. See nvmlDeviceResetApplicationsClocks.

nvmlReturn_t nvmlDeviceResetApplicationsClocks (nvmlDevice_t device)

Parameters

device
The identifier of the target device

Returns

- NVML_SUCCESS if new settings were successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Resets the application clock to the default value

This is the applications clock that will be used after system reboot or driver reload. Default value is constant, but the current value an be changed using nvmlDeviceSetApplicationsClocks.

On Pascal and newer hardware, if clocks were previously locked with nvmlDeviceSetApplicationsClocks, this call will unlock clocks. This returns clocks their default behavior ofautomatically boosting above base clocks as thermal limits allow.

See also:

nvmlDeviceGetApplicationsClock

nvmlDeviceSetApplicationsClocks

For Fermi or newer non-GeForce fully supported devices and Maxwell or newer GeForce devices.

nvmlReturn_t nvmlDeviceSetAutoBoostedClocksEnabled (nvmlDevice_t device, nvmlEnableState_t enabled)

Parameters

device
The identifier of the target device

enabled
What state to try to set Auto Boosted clocks of the target device to

Returns

- NVML_SUCCESS If the Auto Boosted clocks were successfully set to the state specified by enabled
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support Auto Boosted clocks
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Try to set the current state of Auto Boosted clocks on a device.

For Kepler or newer fully supported devices.

Auto Boosted clocks are enabled by default on some hardware, allowing the GPU to run at higher clock rates to maximize performance as thermal limits allow. Auto Boosted clocks should be disabled if fixed clock rates are desired.

Non-root users may use this API by default but can be restricted by root from using this API by calling nvmlDeviceSetAPIRestriction with apiType=NVML_RESTRICTED_API_SET_AUTO_BOOSTED_CLOCKS. Note: Persistence Mode is required to modify current Auto Boost settings, therefore, it must be enabled.

On Pascal and newer hardware, Auto Boosted clocks are controlled through application clocks. Use nvmlDeviceSetApplicationsClocks and nvmlDeviceResetApplicationsClocks to control Auto Boost behavior.

nvmlReturn_t nvmlDeviceSetDefaultAutoBoostedClocksEnabled (nvmlDevice_t device, nvmlEnableState_t enabled, unsigned int flags)

Parameters

device
The identifier of the target device

enabled
What state to try to set default Auto Boosted clocks of the target device to

flags
Flags that change the default behavior. Currently Unused.

Returns

- NVML_SUCCESS If the Auto Boosted clock's default state was successfully set to the state specified by enabled
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NO_PERMISSION If the calling user does not have permission to change Auto Boosted clock's default state.
- NVML_ERROR_INVALID_ARGUMENT if device is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support Auto Boosted clocks
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Try to set the default state of Auto Boosted clocks on a device. This is the default state that Auto Boosted clocks will return to when no compute running processes (e.g. CUDA application which have an active context) are running

For Kepler or newer non-GeForce fully supported devices and Maxwell or newer GeForce devices. Requires root/admin permissions.

Auto Boosted clocks are enabled by default on some hardware, allowing the GPU to run at higher clock rates to maximize performance as thermal limits allow. Auto Boosted clocks should be disabled if fixed clock rates are desired.

On Pascal and newer hardware, Auto Boosted clocks are controlled through application clocks. Use nvmlDeviceSetApplicationsClocks and nvmlDeviceResetApplicationsClocks to control Auto Boost behavior.

nvmlReturn_t nvmlDeviceSetDefaultFanSpeed_v2 (nvmlDevice_t device, unsigned int fan)

Parameters

device
The identifier of the target device

fan
The index of the fan, starting at zero

Description

Sets the speed of the fan control policy to default.

For all cuda-capable discrete products with fans

return NVML_SUCCESS if speed has been adjusted NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized NVML_ERROR_INVALID_ARGUMENT if device is invalid NVML_ERROR_NOT_SUPPORTED if the device does not support this (doesn't have fans) NVML_ERROR_UNKNOWN on any unexpected error

nvmlReturn_t nvmlDeviceSetFanControlPolicy (nvmlDevice_t device, unsigned int fan, nvmlFanControlPolicy_t policy)

Description

Sets current fan control policy.

For Maxwell or newer fully supported devices.

Requires privileged user.

For all cuda-capable discrete products with fans

device The identifier of the target device policy The fan control policy to set

return NVML_SUCCESS if policy has been set NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized NVML_ERROR_INVALID_ARGUMENT if device is invalid or policy is null or the fan given doesn't reference a fan that exists. NVML_ERROR_NOT_SUPPORTED if the device is older than Maxwell NVML_ERROR_UNKNOWN on any unexpected error

nvmlReturn_t nvmlDeviceSetTemperatureThreshold (nvmlDevice_t device, nvmlTemperatureThresholds_t thresholdType, int *temp)

Parameters

device
The identifier of the target device

thresholdType
The type of threshold value to be set

temp
Reference which hold the value to be set

Returns

- NVML_SUCCESS if temp has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, thresholdType is invalid or temp is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not have a temperature sensor or is unsupported
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Sets the temperature threshold for the GPU with the specified threshold type in degrees C.

For Maxwell or newer fully supported devices.

See nvmlTemperatureThresholds_t for details on available temperature thresholds.

nvmlReturn_t nvmlDeviceSetPowerManagementLimit (nvmlDevice_t device, unsigned int limit)

Parameters

device
The identifier of the target device

limit
Power management limit in milliwatts to set

Returns

- NVML_SUCCESS if limit has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or defaultLimit is out of range
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set new power limit of this device.

For Kepler or newer fully supported devices. Requires root/admin permissions.

See nvmlDeviceGetPowerManagementLimitConstraints to check the allowed ranges of values.

Limit is not persistent across reboots or driver unloads. Enable persistent mode to prevent driver from unloading when no application is using the device.

See also:

nvmlDeviceGetPowerManagementLimitConstraints

nvmlDeviceGetPowerManagementDefaultLimit

nvmlReturn_t nvmlDeviceSetGpuOperationMode (nvmlDevice_t device, nvmlGpuOperationMode_t mode)

Parameters

device
The identifier of the target device

mode
Target GOM

Returns

- NVML_SUCCESS if mode has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or mode incorrect

- NVML_ERROR_NOT_SUPPORTED if the device does not support GOM or specific mode
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Sets new GOM. See nvmlGpuOperationMode_t for details.

For GK110 M-class and X-class Tesla products from the Kepler family. Modes NVML_GOM_LOW_DP and NVML_GOM_ALL_ON are supported on fully supported GeForce products. Not supported on Quadro and Tesla C-class products. Requires root/admin permissions.

Changing GOMs requires a reboot. The reboot requirement might be removed in the future.

Compute only GOMs don't support graphics acceleration. Under windows switching to these GOMs when pending driver model is WDDM is not supported. See nvmlDeviceSetDriverModel.

See also:

nvmlGpuOperationMode_t

nvmlDeviceGetGpuOperationMode

nvmlReturn_t nvmlDeviceSetAPIRestriction (nvmlDevice_t device, nvmlRestrictedAPI_t apiType, nvmlEnableState_t isRestricted)

Parameters

device
The identifier of the target device
apiType
Target API type for this operation
isRestricted
The target restriction

Returns

- NVML_SUCCESS if isRestricted has been set

- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or apiType incorrect
- NVML_ERROR_NOT_SUPPORTED if the device does not support changing API restrictions or the device does not support the feature that api restrictions are being set for (E.G. Enabling/disabling auto boosted clocks is not supported by the device)
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Changes the root/admin restructions on certain APIs. See nvmlRestrictedAPI_t for the list of supported APIs. This method can be used by a root/admin user to give non-root/admin access to certain otherwise-restricted APIs. The new setting lasts for the lifetime of the NVIDIA driver; it is not persistent. See nvmlDeviceGetAPIRestriction to query the current restriction settings.

For Kepler or newer fully supported devices. Requires root/admin permissions.

See also:

nvmlRestrictedAPI_t

nvmlReturn_t nvmlDeviceSetFanSpeed_v2 (nvmlDevice_t device, unsigned int fan, unsigned int speed)

Description

Sets the speed of a specified fan.

WARNING: This function changes the fan control policy to manual. It means that YOU have to monitor the temperature and adjust the fan speed accordingly. If you set the fan speed too low you can burn your GPU! Use nvmlDeviceSetDefaultFanSpeed_v2 to restore default control policy.

For all cuda-capable discrete products with fans that are Maxwell or Newer.

device The identifier of the target device fan The index of the fan, starting at zero speed The target speed of the fan [0-100] in % of max speed

return NVML_SUCCESS if the fan speed has been set
NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
NVML_ERROR_INVALID_ARGUMENT if the device is not valid, or the speed

is outside acceptable ranges, or if the fan index doesn't reference an actual fan. NVML_ERROR_NOT_SUPPORTED if the device is older than Maxwell. NVML_ERROR_UNKNOWN if there was an unexpected error.

nvmlReturn_t nvmlDeviceSetGpcClkVfOffset (nvmlDevice_t device, int offset)

Parameters

device
The identifier of the target device

offset
The GPCCLK VF offset value to set

Returns

- NVML_SUCCESS if offset has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or offset is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set the GPCCLK VF offset value

nvmlReturn_t nvmlDeviceSetMemClkVfOffset (nvmlDevice_t device, int offset)

Parameters

device
The identifier of the target device

offset
The MemClk VF offset value to set

Returns

- NVML_SUCCESS if offset has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or offset is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature

- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set the MemClk (Memory Clock) VF offset value. It requires elevated privileges.

nvmlReturn_t nvmlDeviceSetConfComputeUnprotectedMemSize (nvmlDevice_t device, unsigned long long sizeKiB)

Parameters

device
Device Handle
sizeKiB
Unprotected Memory size to be set in KiB

Returns

- NVML_SUCCESS if sizeKiB successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device

Description

Set Conf Computing Unprotected Memory Size.

For Ampere or newer fully supported devices. Supported on Linux, Windows TCC.

nvmlReturn_t nvmlSystemSetConfComputeGpusReadyState (unsigned int isAcceptingWork)

Parameters

isAcceptingWork
GPU accepting new work, NVML_CC_ACCEPTING_CLIENT_REQUESTS_TRUE or NVML_CC_ACCEPTING_CLIENT_REQUESTS_FALSE

Description

Set Conf Computing GPUs ready state.

For Ampere or newer fully supported devices. Supported on Linux, Windows TCC.

return

- NVML_SUCCESS if current GPUs ready state is successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if isAcceptingWork is invalid
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device

nvmlReturn_t nvmlSystemSetConfComputeKeyRotationThresholdInfo (nvmlConfComputeSetKeyRotationThresholdInfo_t *pKeyRotationThrInfo)

Parameters

pKeyRotationThrInfo
Reference to the key rotation threshold data

Returns

- NVML_SUCCESS if key rotation threashold max attacker advantage has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or memory is NULL
- NVML_ERROR_INVALID_STATE if confidential compute GPU ready state is enabled
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set Conf Computing key rotation threshold.

HOPPER_OR_NEWER% Supported on Linux, Windows TCC.

This function is to set the confidential compute key rotation threshold parameters. pKeyRotationThrInfo->maxAttackerAdvantage should be in the range from NVML_CC_KEY_ROTATION_THRESHOLD_ATTACKER_ADVANTAGE_MIN to NVML_CC_KEY_ROTATION_THRESHOLD_ATTACKER_ADVANTAGE_MAX. Default value is 60.

4.18. NvLink Methods

This chapter describes methods that NVML can perform on NVLINK enabled devices.

nvmlReturn_t nvmlDeviceGetNvLinkState (nvmlDevice_t device, unsigned int link, nvmlEnableState_t *isActive)

Parameters

device
The identifier of the target device

link
Specifies the NvLink link to be queried

isActive
nvmlEnableState_t where NVML_FEATURE_ENABLED indicates that the link is active and NVML_FEATURE_DISABLED indicates it is inactive

Returns

- NVML_SUCCESS if isActive has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device or link is invalid or isActive is NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the state of the device's NvLink for the link specified

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetNvLinkVersion (nvmlDevice_t device, unsigned int link, unsigned int *version)

Parameters

device
The identifier of the target device

link
Specifies the NvLink link to be queried
version
Requested NvLink version

Returns

- NVML_SUCCESS if version has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device or link is invalid or version is NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the version of the device's NvLink for the link specified

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetNvLinkCapability (nvmlDevice_t device, unsigned int link, nvmlNvLinkCapability_t capability, unsigned int *capResult)

Parameters

device
The identifier of the target device
link
Specifies the NvLink link to be queried
capability
Specifies the nvmlNvLinkCapability_t to be queried
capResult
A boolean for the queried capability indicating that feature is available

Returns

- NVML_SUCCESS if capResult has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, link, or capability is invalid or capResult is NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature

- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the requested capability from the device's NvLink for the link specified Please refer to the nvmlNvLinkCapability_t structure for the specific caps that can be queried The return value should be treated as a boolean.

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetNvLinkRemotePciInfo_v2 (nvmlDevice_t device, unsigned int link, nvmlPciInfo_t *pci)

Parameters

device
The identifier of the target device

link
Specifies the NvLink link to be queried

pci
nvmlPciInfo_t of the remote node for the specified link

Returns

- NVML_SUCCESS if pci has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device or link is invalid or pci is NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the PCI information for the remote node on a NvLink link Note: pciSubSystemId is not filled in this function and is indeterminate

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetNvLinkErrorCounter (nvmlDevice_t device, unsigned int link,

nvmlNvLinkErrorCounter_t counter, unsigned long long *counterValue)

Parameters

device
The identifier of the target device
link
Specifies the NvLink link to be queried
counter
Specifies the NvLink counter to be queried
counterValue
Returned counter value

Returns

- NVML_SUCCESS if counter has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, link, or counter is invalid or counterValue is NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the specified error counter value Please refer to nvmlNvLinkErrorCounter_t for error counters that are available

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceResetNvLinkErrorCounters (nvmlDevice_t device, unsigned int link)

Parameters

device
The identifier of the target device
link
Specifies the NvLink link to be queried

Returns

- NVML_SUCCESS if the reset is successful
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if device or link is invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Resets all error counters to zero Please refer to nvmlNvLinkErrorCounter_t for the list of error counters that are reset

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceSetNvLinkUtilizationControl (nvmlDevice_t device, unsigned int link, unsigned int counter, nvmlNvLinkUtilizationControl_t *control, unsigned int reset)

Parameters

device
The identifier of the target device
link
Specifies the NvLink link to be queried
counter
Specifies the counter that should be set (0 or 1).
control
A reference to the nvmlNvLinkUtilizationControl_t to set
reset
Resets the counters on set if non-zero

Returns

- NVML_SUCCESS if the control has been set successfully
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, counter, link, or control is invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Deprecated: Setting utilization counter control is no longer supported.

Set the NVLINK utilization counter control information for the specified counter, 0 or 1. Please refer to nvmlNvLinkUtilizationControl_t for the structure definition. Performs a reset of the counters if the reset parameter is non-zero.

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetNvLinkUtilizationControl (nvmlDevice_t device, unsigned int link, unsigned int counter, nvmlNvLinkUtilizationControl_t *control)

Parameters

device
The identifier of the target device
link
Specifies the NvLink link to be queried
counter
Specifies the counter that should be set (0 or 1).
control
A reference to the nvmlNvLinkUtilizationControl_t to place information

Returns

- NVML_SUCCESS if the control has been set successfully
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, counter, link, or control is invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Deprecated: Getting utilization counter control is no longer supported.

Get the NVLINK utilization counter control information for the specified counter, 0 or 1. Please refer to nvmlNvLinkUtilizationControl_t for the structure definition

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetNvLinkUtilizationCounter (nvmlDevice_t device, unsigned int link, unsigned int

counter, unsigned long long *rxcounter, unsigned long long *txcounter)

Parameters

device
The identifier of the target device
link
Specifies the NvLink link to be queried
counter
Specifies the counter that should be read (0 or 1).
rxcounter
Receive counter return value
txcounter
Transmit counter return value

Returns

- NVML_SUCCESS if rxcounter and txcounter have been successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, counter, or link is invalid or rxcounter or txcounter are NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Deprecated: Use nvmlDeviceGetFieldValues with NVML_FI_DEV_NVLINK_THROUGHPUT_* as field values instead.

Retrieve the NVLINK utilization counter based on the current control for a specified counter. In general it is good practice to use nvmlDeviceSetNvLinkUtilizationControl before reading the utilization counters as they have no default state

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceFreezeNvLinkUtilizationCounter

(nvmlDevice_t device, unsigned int link, unsigned int counter, nvmlEnableState_t freeze)

Parameters

device
The identifier of the target device
link
Specifies the NvLink link to be queried
counter
Specifies the counter that should be frozen (0 or 1).
freeze
NVML_FEATURE_ENABLED = freeze the receive and transmit counters
NVML_FEATURE_DISABLED = unfreeze the receive and transmit counters

Returns

- NVML_SUCCESS if counters were successfully frozen or unfrozen
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, link, counter, or freeze is invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Deprecated: Freezing NVLINK utilization counters is no longer supported.

Freeze the NVLINK utilization counters Both the receive and transmit counters are operated on by this function

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceResetNvLinkUtilizationCounter (nvmlDevice_t device, unsigned int link, unsigned int counter)

Parameters

device
The identifier of the target device
link
Specifies the NvLink link to be reset
counter
Specifies the counter that should be reset (0 or 1)

Returns

- NVML_SUCCESS if counters were successfully reset
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, link, or counter is invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Deprecated: Resetting NVLINK utilization counters is no longer supported.

Reset the NVLINK utilization counters Both the receive and transmit counters are operated on by this function

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetNvLinkRemoteDeviceType (nvmlDevice_t device, unsigned int link, nvmlIntNvLinkDeviceType_t *pNvLinkDeviceType)

Parameters

device
The device handle of the target GPU
link
The NVLink link index on the target GPU
pNvLinkDeviceType
Pointer in which the output remote device type is returned

Returns

- NVML_SUCCESS if pNvLinkDeviceType has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_SUPPORTED if NVLink is not supported
- NVML_ERROR_INVALID_ARGUMENT if device or link is invalid, or pNvLinkDeviceType is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get the NVLink device type of the remote device connected over the given link.

4.19. Event Handling Methods

This chapter describes methods that NVML can perform against each device to register and wait for some event to occur.

struct nvmlEventData_t

Event Types

typedef struct nvmlEventSet_st *nvmlEventSet_t

Handle to an event set

nvmlReturn_t nvmlEventSetCreate (nvmlEventSet_t *set)

Parameters

set
Reference in which to return the event handle

Returns

- NVML_SUCCESS if the event has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if set is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Create an empty set of events. Event set should be freed by nvmlEventSetFree

For Fermi or newer fully supported devices.

See also:

nvmlEventSetFree

nvmlReturn_t nvmlDeviceRegisterEvents (nvmlDevice_t device, unsigned long long eventTypes, nvmlEventSet_t set)

Parameters

device
The identifier of the target device
eventTypes
Bitmask of Event Types to record
set
Set to which add new event types

Returns

- NVML_SUCCESS if the event has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if eventTypes is invalid or set is NULL
- NVML_ERROR_NOT_SUPPORTED if the platform does not support this feature or some of requested event types
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Starts recording of events on a specified devices and add the events to specified nvmlEventSet_t

For Fermi or newer fully supported devices. Ecc events are available only on ECC enabled devices (see nvmlDeviceGetTotalEccErrors) Power capping events are available only on Power Management enabled devices (see nvmlDeviceGetPowerManagementMode)

For Linux only.

IMPORTANT: Operations on set are not thread safe

This call starts recording of events on specific device. All events that occurred before this call are not recorded. Checking if some event occurred can be done with nvmlEventSetWait_v2

If function reports NVML_ERROR_UNKNOWN, event set is in undefined state and should be freed. If function reports NVML_ERROR_NOT_SUPPORTED, event set can still be used. None of the requested eventTypes are registered in that case.

See also:

Event Types

nvmlDeviceGetSupportedEventTypes

nvmlEventSetWait

nvmlEventSetFree

nvmlReturn_t nvmlDeviceGetSupportedEventTypes (nvmlDevice_t device, unsigned long long *eventTypes)

Parameters

device
The identifier of the target device

eventTypes
Reference in which to return bitmask of supported events

Returns

- NVML_SUCCESS if the eventTypes has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if eventType is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Returns information about events supported on device

For Fermi or newer fully supported devices.

Events are not supported on Windows. So this function returns an empty mask in eventTypes on Windows.

See also:

Event Types

nvmlDeviceRegisterEvents

nvmlReturn_t nvmlEventSetWait_v2 (nvmlEventSet_t set, nvmlEventData_t *data, unsigned int timeoutms)

Parameters

set
Reference to set of events to wait on

data
Reference in which to return event data

timeoutms
Maximum amount of wait time in milliseconds for registered event

Returns

- NVML_SUCCESS if the data has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if data is NULL
- NVML_ERROR_TIMEOUT if no event arrived in specified timeout or interrupt arrived
- NVML_ERROR_GPU_IS_LOST if a GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Waits on events and delivers events

For Fermi or newer fully supported devices.

If some events are ready to be delivered at the time of the call, function returns immediately. If there are no events ready to be delivered, function sleeps till event arrives but not longer than specified timeout. This function in certain conditions can return before specified timeout passes (e.g. when interrupt arrives)

On Windows, in case of xid error, the function returns the most recent xid error type seen by the system. If there are multiple xid errors generated before nvmlEventSetWait is invoked then the last seen xid error type is returned for all xid error events.

On Linux, every xid error event would return the associated event data and other information if applicable.

In MIG mode, if device handle is provided, the API reports all the events for the available instances, only if the caller has appropriate privileges. In absence of required privileges, only the events which affect all the instances (i.e. whole device) are reported.

This API does not currently support per-instance event reporting using MIG device handles.

See also:

Event Types

nvmlDeviceRegisterEvents

nvmlReturn_t nvmlEventSetFree (nvmlEventSet_t set)

Parameters

set
Reference to events to be released

Returns

- NVML_SUCCESS if the event has been successfully released
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Releases events in the set

For Fermi or newer fully supported devices.

See also:

nvmlDeviceRegisterEvents

4.19.1. Event Types

Event Handling Methods

Event Types which user can be notified about. See description of particular functions for details.

See nvmlDeviceRegisterEvents and nvmlDeviceGetSupportedEventTypes to check which devices support each event.

Types can be combined with bitwise or operator '|' when passed to nvmlDeviceRegisterEvents

#define nvmlEventTypeSingleBitEccError 0x0000000000000001LL

Event about single bit ECC errors.

A corrected texture memory error is not an ECC error, so it does not generate a single bit event

#define nvmlEventTypeDoubleBitEccError 0x0000000000000002LL

Event about double bit ECC errors.

An uncorrected texture memory error is not an ECC error, so it does not generate a double bit event

#define nvmlEventTypePState 0x0000000000000004LL

Event about PState changes.

On Fermi architecture PState changes are also an indicator that GPU is throttling down due to no work being executed on the GPU, power capping or thermal capping. In a typical situation, Fermi-based GPU should stay in P0 for the duration of the execution of the compute process.

#define nvmlEventTypeXidCriticalError 0x0000000000000008LL

Event that Xid critical error occurred.

#define nvmlEventTypeClock 0x0000000000000010LL

Event about clock changes.

Kepler only

#define nvmlEventTypePowerSourceChange 0x0000000000000080LL

Event about AC/Battery power source changes.

#define nvmlEventMigConfigChange 0x0000000000000100LL

Event about MIG configuration changes.

#define nvmlEventTypeNone 0x0000000000000000LL

Mask with no events.

#define nvmlEventTypeAll (nvmlEventTypeNone \ | nvmlEventTypeSingleBitEccError \ | nvmlEventTypeDoubleBitEccError \ | nvmlEventTypePState \ | nvmlEventTypeClock \ | nvmlEventTypeXidCriticalError \ | nvmlEventTypePowerSourceChange \ | nvmlEventMigConfigChange \ )

Mask of all events.

4.20. Drain states

This chapter describes methods that NVML can perform against each device to control their drain state and recognition by NVML and NVIDIA kernel driver. These methods can be used with out-of-band tools to power on/off GPUs, enable robust reset scenarios, etc.

nvmlReturn_t nvmlDeviceModifyDrainState (nvmlPciInfo_t *pciInfo, nvmlEnableState_t newState)

Parameters

pciInfo
The PCI address of the GPU drain state to be modified

newState
The drain state that should be entered, see nvmlEnableState_t

Returns

- NVML_SUCCESS if counters were successfully reset
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if nvmlIndex or newState is invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_NO_PERMISSION if the calling process has insufficient permissions to perform operation
- NVML_ERROR_IN_USE if the device has persistence mode turned on
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Modify the drain state of a GPU. This method forces a GPU to no longer accept new incoming requests. Any new NVML process will no longer see this GPU. Persistence

mode for this GPU must be turned off before this call is made. Must be called as administrator. For Linux only.

For Pascal or newer fully supported devices. Some Kepler devices supported.

nvmlReturn_t nvmlDeviceQueryDrainState (nvmlPciInfo_t *pciInfo, nvmlEnableState_t *currentState)

Parameters

pciInfo
The PCI address of the GPU drain state to be queried

currentState
The current drain state for this GPU, see nvmlEnableState_t

Returns

- NVML_SUCCESS if counters were successfully reset
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if nvmlIndex or currentState is invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Query the drain state of a GPU. This method is used to check if a GPU is in a currently draining state. For Linux only.

For Pascal or newer fully supported devices. Some Kepler devices supported.

nvmlReturn_t nvmlDeviceRemoveGpu_v2 (nvmlPciInfo_t *pciInfo, nvmlDetachGpuState_t gpuState, nvmlPcieLinkState_t linkState)

Parameters

pciInfo
The PCI address of the GPU to be removed

gpuState
Whether the GPU is to be removed, from the OS see nvmlDetachGpuState_t

linkState
Requested upstream PCIe link state, see nvmlPcieLinkState_t

Returns

- NVML_SUCCESS if counters were successfully reset
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if nvmlIndex is invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_IN_USE if the device is still in use and cannot be removed

Description

This method will remove the specified GPU from the view of both NVML and the NVIDIA kernel driver as long as no other processes are attached. If other processes are attached, this call will return NVML_ERROR_IN_USE and the GPU will be returned to its original "draining" state. Note: the only situation where a process can still be attached after nvmlDeviceModifyDrainState() is called to initiate the draining state is if that process was using, and is still using, a GPU before the call was made. Also note, persistence mode counts as an attachment to the GPU thus it must be disabled prior to this call.

For long-running NVML processes please note that this will change the enumeration of current GPUs. For example, if there are four GPUs present and GPU1 is removed, the new enumeration will be 0-2. Also, device handles after the removed GPU will not be valid and must be re-established. Must be run as administrator. For Linux only.

For Pascal or newer fully supported devices. Some Kepler devices supported.

nvmlReturn_t nvmlDeviceDiscoverGpus (nvmlPciInfo_t *pciInfo)

Parameters

pciInfo
The PCI tree to be searched. Only the domain, bus, and device fields are used in this call.

Returns

- NVML_SUCCESS if counters were successfully reset
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if pciInfo is invalid
- NVML_ERROR_NOT_SUPPORTED if the operating system does not support this feature
- NVML_ERROR_OPERATING_SYSTEM if the operating system is denying this feature

- NVML_ERROR_NO_PERMISSION if the calling process has insufficient permissions to perform operation
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Request the OS and the NVIDIA kernel driver to rediscover a portion of the PCI subsystem looking for GPUs that were previously removed. The portion of the PCI tree can be narrowed by specifying a domain, bus, and device. If all are zeroes then the entire PCI tree will be searched. Please note that for long-running NVML processes the enumeration will change based on how many GPUs are discovered and where they are inserted in bus order.

In addition, all newly discovered GPUs will be initialized and their ECC scrubbed which may take several seconds per GPU. Also, all device handles are no longer guaranteed to be valid post discovery.

Must be run as administrator. For Linux only.

For Pascal or newer fully supported devices. Some Kepler devices supported.

4.21. Field Value Queries

This chapter describes NVML operations that are associated with retrieving Field Values from NVML

nvmlReturn_t nvmlDeviceGetFieldValues (nvmlDevice_t device, int valuesCount, nvmlFieldValue_t *values)

Parameters

device
The device handle of the GPU to request field values for

valuesCount
Number of entries in values that should be retrieved

values
Array of valuesCount structures to hold field values. Each value's fieldId must be populated prior to this call

Returns

- NVML_SUCCESS if any values in values were populated. Note that you must check the nvmlReturn field of each value for each individual status
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or values is NULL

Description

Request values for a list of fields for a device. This API allows multiple fields to be queried at once. If any of the underlying fieldIds are populated by the same driver call, the results for those field IDs will be populated from a single call rather than making a driver call for each fieldId.

nvmlReturn_t nvmlDeviceClearFieldValues (nvmlDevice_t device, int valuesCount, nvmlFieldValue_t *values)

Parameters

device
The device handle of the GPU to request field values for

valuesCount
Number of entries in values that should be cleared

values
Array of valuesCount structures to hold field values. Each value's fieldId must be populated prior to this call

Returns

- NVML_SUCCESS if any values in values were cleared. Note that you must check the nvmlReturn field of each value for each individual status
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or values is NULL

Description

Clear values for a list of fields for a device. This API allows multiple fields to be cleared at once.

4.22. Enums, Constants and Structs

4.23. vGPU APIs

This chapter describes operations that are associated with NVIDIA vGPU Software products.

nvmlReturn_t nvmlDeviceGetVirtualizationMode (nvmlDevice_t device, nvmlGpuVirtualizationMode_t *pVirtualMode)

Parameters

device
Identifier of the target device

pVirtualMode
Reference to virtualization mode. One of NVML_GPU_VIRTUALIZATION_?

Returns

- NVML_SUCCESS if pVirtualMode is fetched
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or pVirtualMode is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

This method is used to get the virtualization mode corresponding to the GPU.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetHostVgpuMode (nvmlDevice_t device, nvmlHostVgpuMode_t *pHostVgpuMode)

Parameters

device
The identifier of the target device

pHostVgpuMode
Reference in which to return the current vGPU mode

Returns

- NVML_SUCCESS if device's vGPU mode has been successfully retrieved
- NVML_ERROR_INVALID_ARGUMENT if device handle is 0 or pVgpuMode is NULL

- NVML_ERROR_NOT_SUPPORTED if device doesn't support this feature.
- NVML_ERROR_UNKNOWN if any unexpected error occurred

Description

Queries if SR-IOV host operation is supported on a vGPU supported device.

Checks whether SR-IOV host capability is supported by the device and the driver, and indicates device is in SR-IOV mode if both of these conditions are true.

nvmlReturn_t nvmlDeviceSetVirtualizationMode (nvmlDevice_t device, nvmlGpuVirtualizationMode_t virtualMode)

Parameters

device
Identifier of the target device

virtualMode
virtualization mode. One of NVML_GPU_VIRTUALIZATION_?

Returns

- NVML_SUCCESS if virtualMode is set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or virtualMode is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_NOT_SUPPORTED if setting of virtualization mode is not supported.
- NVML_ERROR_NO_PERMISSION if setting of virtualization mode is not allowed for this client.

Description

This method is used to set the virtualization mode corresponding to the GPU.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetVgpuHeterogeneousMode (nvmlDevice_t device, nvmlVgpuHeterogeneousMode_t *pHeterogeneousMode)

Parameters

device
The identifier of the target device

pHeterogeneousMode
Pointer to the caller-provided structure of nvmlVgpuHeterogeneousMode_t

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device is invalid or pHeterogeneousMode is NULL
- NVML_ERROR_NOT_SUPPORTED If device doesn't support this feature
- NVML_ERROR_VERSION_MISMATCH If the version of pHeterogeneousMode is invalid
- NVML_ERROR_UNKNOWN On any unexpected error

Description

Get the vGPU heterogeneous mode for the device.

When in heterogeneous mode, a vGPU can concurrently host timesliced vGPUs with differing framebuffer sizes.

On successful return, the function returns pHeterogeneousMode->mode with the current vGPU heterogeneous mode. pHeterogeneousMode->version is the version number of the structure nvmlVgpuHeterogeneousMode_t, the caller should set the correct version number to retrieve the vGPU heterogeneous mode. pHeterogeneousMode->mode can either be NVML_FEATURE_ENABLED or NVML_FEATURE_DISABLED.

nvmlReturn_t nvmlDeviceSetVgpuHeterogeneousMode (nvmlDevice_t device, const nvmlVgpuHeterogeneousMode_t *pHeterogeneousMode)

Parameters

device
Identifier of the target device

pHeterogeneousMode
Pointer to the caller-provided structure of nvmlVgpuHeterogeneousMode_t

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device or pHeterogeneousMode is NULL or pHeterogeneousMode->mode is invalid
- NVML_ERROR_IN_USE If the device is in use
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation
- NVML_ERROR_NOT_SUPPORTED If MIG is enabled or device doesn't support this feature
- NVML_ERROR_VERSION_MISMATCH If the version of pHeterogeneousMode is invalid
- NVML_ERROR_UNKNOWN On any unexpected error

Description

Enable or disable vGPU heterogeneous mode for the device.

When in heterogeneous mode, a vGPU can concurrently host timesliced vGPUs with differing framebuffer sizes.

API would return an appropriate error code upon unsuccessful activation. For example, the heterogeneous mode set will fail with error NVML_ERROR_IN_USE if any vGPU instance is active on the device. The caller of this API is expected to shutdown the vGPU VMs and retry setting the mode. On successful return, the function updates the vGPU heterogeneous mode with the user provided pHeterogeneousMode->mode. pHeterogeneousMode->version is the version number of the structure nvmlVgpuHeterogeneousMode_t, the caller should set the correct version number to set the vGPU heterogeneous mode.

nvmlReturn_t nvmlVgpuInstanceGetPlacementId (nvmlVgpuInstance_t vgpuInstance, nvmlVgpuPlacementId_t *pPlacement)

Parameters

vgpuInstance
Identifier of the target vGPU instance

pPlacement
Pointer to vGPU placement ID structure nvmlVgpuPlacementId_t

Returns

- NVML_SUCCESS If information is successfully retrieved
- NVML_ERROR_NOT_FOUND If vgpuInstance does not match a valid active vGPU instance
- NVML_ERROR_INVALID_ARGUMENT If vgpuInstance is invalid or pPlacement is NULL
- NVML_ERROR_VERSION_MISMATCH If the version of pPlacement is invalid
- NVML_ERROR_UNKNOWN On any unexpected error

Description

Query the placement ID of active vGPU instance.

When in vGPU heterogeneous mode, this function returns a valid placement ID as pPlacement->placementId else NVML_INVALID_VGPU_PLACEMENT_ID is returned. pPlacement->version is the version number of the structure nvmlVgpuPlacementId_t, the caller should set the correct version number to get placement id of the vGPU instance vgpuInstance.

nvmlReturn_t nvmlDeviceGetVgpuTypeSupportedPlacements (nvmlDevice_t device, nvmlVgpuTypeId_t vgpuTypeId, nvmlVgpuPlacementList_t *pPlacementList)

Parameters

device
Identifier of the target device

vgpuTypeId
Handle to vGPU type. The vGPU type ID

pPlacementList
Pointer to the vGPU placement structure nvmlVgpuPlacementList_t

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device or vgpuTypeId is invalid or pPlacementList is NULL
- NVML_ERROR_NOT_SUPPORTED If device or vgpuTypeId isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation
- NVML_ERROR_VERSION_MISMATCH If the version of pPlacementList is invalid
- NVML_ERROR_UNKNOWN On any unexpected error

Description

Query the supported vGPU placement ID of the vGPU type.

An array of supported vGPU placement IDs for the vGPU type ID indicated by vgpuTypeId is returned in the caller-supplied buffer of pPlacementList->placementIds. Memory needed for the placementIds array should be allocated based on maximum instances of a vGPU type which can be queried via nvmlVgpuTypeGetMaxInstances().

This function will return supported placement IDs even if GPU is not in vGPU heterogeneous mode.

nvmlReturn_t nvmlDeviceGetVgpuTypeCreatablePlacements (nvmlDevice_t device, nvmlVgpuTypeId_t vgpuTypeId, nvmlVgpuPlacementList_t *pPlacementList)

Parameters

device
The identifier of the target device
vgpuTypeId
Handle to vGPU type. The vGPU type ID
pPlacementList
Pointer to the list of vGPU placement structure nvmlVgpuPlacementList_t

Returns

- NVML_SUCCESS Upon success

- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device or vgpuTypeId is invalid or pPlacementList is NULL
- NVML_ERROR_NOT_SUPPORTED If device or vgpuTypeId isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation
- NVML_ERROR_VERSION_MISMATCH If the version of pPlacementList is invalid
- NVML_ERROR_UNKNOWN On any unexpected error

Description

Query the creatable vGPU placement ID of the vGPU type.

An array of creatable vGPU placement IDs for the vGPU type ID indicated by vgpuTypeId is returned in the caller-supplied buffer of pPlacementList->placementIds. Memory needed for the placementIds array should be allocated based on maximum instances of a vGPU type which can be queried via nvmlVgpuTypeGetMaxInstances(). The creatable vGPU placement IDs may differ over time, as there may be restrictions on what type of vGPU the vGPU instance is running.

The function will return NVML_ERROR_NOT_SUPPORTED if the device is not in vGPU heterogeneous mode.

nvmlReturn_t nvmlVgpuTypeGetGspHeapSize (nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *gspHeapSize)

Parameters

vgpuTypeId
Handle to vGPU type

gspHeapSize
Reference to return the GSP heap size value

Returns

- NVML_SUCCESS Successful completion
- NVML_ERROR_UNINITIALIZED If the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If vgpuTypeId is invalid, or gspHeapSize is NULL
- NVML_ERROR_UNKNOWN On any unexpected error

Description

Retrieve the static GSP heap size of the vGPU type in bytes

nvmlReturn_t nvmlVgpuTypeGetFbReservation (nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *fbReservation)

Parameters

vgpuTypeId
Handle to vGPU type

fbReservation
Reference to return the framebuffer reservation

Returns

- NVML_SUCCESS Successful completion
- NVML_ERROR_UNINITIALIZED If the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If vgpuTypeId is invalid, or fbReservation is NULL
- NVML_ERROR_UNKNOWN On any unexpected error

Description

Retrieve the static framebuffer reservation of the vGPU type in bytes

nvmlReturn_t nvmlDeviceSetVgpuCapabilities (nvmlDevice_t device, nvmlDeviceVgpuCapability_t capability, nvmlEnableState_t state)

Parameters

device
The identifier of the target device

capability
Specifies the nvmlDeviceVgpuCapability_t to be set

state
The target capability mode

Returns

- NVML_SUCCESS Successful completion

- NVML_ERROR_UNINITIALIZED If the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device is invalid, or capability is invalid, or state is invalid
- NVML_ERROR_NOT_SUPPORTED The API is not supported in current state, or device not in vGPU mode
- NVML_ERROR_UNKNOWN On any unexpected error

Description

Set the desirable vGPU capability of a device

Refer to the nvmlDeviceVgpuCapability_t structure for the specific capabilities that can be set. See nvmlEnableState_t for available state.

nvmlReturn_t nvmlDeviceGetGridLicensableFeatures_v4 (nvmlDevice_t device, nvmlGridLicensableFeatures_t *pGridLicensableFeatures)

Parameters

device
Identifier of the target device

pGridLicensableFeatures
Pointer to structure in which vGPU software licensable features are returned

Returns

- NVML_SUCCESS if licensable features are successfully retrieved
- NVML_ERROR_INVALID_ARGUMENT if pGridLicensableFeatures is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the vGPU Software licensable features.

Identifies whether the system supports vGPU Software Licensing. If it does, return the list of licensable feature(s) and their current license status.

4.24. vGPU Management

This chapter describes APIs supporting NVIDIA vGPU.

nvmlReturn_t nvmlGetVgpuDriverCapabilities (nvmlVgpuDriverCapability_t capability, unsigned int *capResult)

Parameters

capability
Specifies the nvmlVgpuDriverCapability_t to be queried

capResult
A boolean for the queried capability indicating that feature is supported

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if capability is invalid, or capResult is NULL
- NVML_ERROR_NOT_SUPPORTED the API is not supported in current state or devices not in vGPU mode
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the requested vGPU driver capability.

Refer to the nvmlVgpuDriverCapability_t structure for the specific capabilities that can be queried. The return value in capResult should be treated as a boolean, with a non-zero value indicating that the capability is supported.

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetVgpuCapabilities (nvmlDevice_t device, nvmlDeviceVgpuCapability_t capability, unsigned int *capResult)

Parameters

device
The identifier of the target device

capability
Specifies the nvmlDeviceVgpuCapability_t to be queried

capResult
Specifies that the queried capability is supported, and also returns capability's data

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or capability is invalid, or capResult is NULL
- NVML_ERROR_NOT_SUPPORTED the API is not supported in current state or device not in vGPU mode
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the requested vGPU capability for GPU.

Refer to the nvmlDeviceVgpuCapability_t structure for the specific capabilities that can be queried. The return value in capResult reports a non-zero value indicating that the capability is supported, and also reports the capability's data based on the queried capability.

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetSupportedVgpus (nvmlDevice_t device, unsigned int *vgpuCount, nvmlVgpuTypeId_t *vgpuTypeIds)

Parameters

device
The identifier of the target device

vgpuCount
Pointer to caller-supplied array size, and returns number of vGPU types

vgpuTypeIds
Pointer to caller-supplied array in which to return list of vGPU types

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_INSUFFICIENT_SIZE vgpuTypeIds buffer is too small, array element count is returned in vgpuCount
- NVML_ERROR_INVALID_ARGUMENT if vgpuCount is NULL or device is invalid
- NVML_ERROR_NOT_SUPPORTED if vGPU is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the supported vGPU types on a physical GPU (device).

An array of supported vGPU types for the physical GPU indicated by device is returned in the caller-supplied buffer pointed at by vgpuTypeIds. The element count of nvmlVgpuTypeId_t array is passed in vgpuCount, and vgpuCount is used to return the number of vGPU types written to the buffer.

If the supplied buffer is not large enough to accommodate the vGPU type array, the function returns NVML_ERROR_INSUFFICIENT_SIZE, with the element count of nvmlVgpuTypeId_t array required in vgpuCount. To query the number of vGPU types supported for the GPU, call this function with *vgpuCount = 0. The code will return NVML_ERROR_INSUFFICIENT_SIZE, or NVML_SUCCESS if no vGPU types are supported.

nvmlReturn_t nvmlDeviceGetCreatableVgpus (nvmlDevice_t device, unsigned int *vgpuCount, nvmlVgpuTypeId_t *vgpuTypeIds)

Parameters

device
The identifier of the target device

vgpuCount
Pointer to caller-supplied array size, and returns number of vGPU types

vgpuTypeIds
Pointer to caller-supplied array in which to return list of vGPU types

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_INSUFFICIENT_SIZE vgpuTypeIds buffer is too small, array element count is returned in vgpuCount
- NVML_ERROR_INVALID_ARGUMENT if vgpuCount is NULL
- NVML_ERROR_NOT_SUPPORTED if vGPU is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the currently creatable vGPU types on a physical GPU (device).

An array of creatable vGPU types for the physical GPU indicated by device is returned in the caller-supplied buffer pointed at by vgpuTypeIds. The element count of

nvmlVgpuTypeId_t array is passed in vgpuCount, and vgpuCount is used to return the number of vGPU types written to the buffer.

The creatable vGPU types for a device may differ over time, as there may be restrictions on what type of vGPU types can concurrently run on a device. For example, if only one vGPU type is allowed at a time on a device, then the creatable list will be restricted to whatever vGPU type is already running on the device.

If the supplied buffer is not large enough to accommodate the vGPU type array, the function returns NVML_ERROR_INSUFFICIENT_SIZE, with the element count of nvmlVgpuTypeId_t array required in vgpuCount. To query the number of vGPU types that can be created for the GPU, call this function with *vgpuCount = 0. The code will return NVML_ERROR_INSUFFICIENT_SIZE, or NVML_SUCCESS if no vGPU types are creatable.

nvmlReturn_t nvmlVgpuTypeGetClass (nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeClass, unsigned int *size)

Parameters

vgpuTypeId
Handle to vGPU type

vgpuTypeClass
Pointer to string array to return class in

size
Size of string

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_INVALID_ARGUMENT if vgpuTypeId is invalid, or vgpuTypeClass is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if size is too small
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the class of a vGPU type. It will not exceed 64 characters in length (including the NUL terminator). See nvmlConstants::NVML_DEVICE_NAME_BUFFER_SIZE.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuTypeGetName (nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeName, unsigned int *size)

Parameters

vgpuTypeId
Handle to vGPU type
vgpuTypeName
Pointer to buffer to return name
size
Size of buffer

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_INVALID_ARGUMENT if vgpuTypeId is invalid, or name is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if size is too small
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the vGPU type name.

The name is an alphanumeric string that denotes a particular vGPU, e.g. GRID M60-2Q. It will not exceed 64 characters in length (including the NUL terminator). See nvmlConstants::NVML_DEVICE_NAME_BUFFER_SIZE.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuTypeGetGpuInstanceProfileId (nvmlVgpuTypeId_t vgpuTypeId, unsigned int *gpuInstanceProfileId)

Parameters

vgpuTypeId
Handle to vGPU type
gpuInstanceProfileId
GPU Instance Profile ID

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_NOT_SUPPORTED if device is not in vGPU Host virtualization mode
- NVML_ERROR_INVALID_ARGUMENT if vgpuTypeId is invalid, or gpuInstanceProfileId is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the GPU Instance Profile ID for the given vGPU type ID. The API will return a valid GPU Instance Profile ID for the MIG capable vGPU types, else INVALID_GPU_INSTANCE_PROFILE_ID is returned.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuTypeGetDeviceID (nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *deviceID, unsigned long long *subsystemID)

Parameters

vgpuTypeId
Handle to vGPU type

deviceID
Device ID and vendor ID of the device contained in single 32 bit value

subsystemID
Subsystem ID and subsystem vendor ID of the device contained in single 32 bit value

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuTypeId is invalid, or deviceId or subsystemID are NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the device ID of a vGPU type.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuTypeGetFramebufferSize (nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *fbSize)

Parameters

vgpuTypeId
Handle to vGPU type
fbSize
Pointer to framebuffer size in bytes

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuTypeId is invalid, or fbSize is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the vGPU framebuffer size in bytes.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuTypeGetNumDisplayHeads (nvmlVgpuTypeId_t vgpuTypeId, unsigned int *numDisplayHeads)

Parameters

vgpuTypeId
Handle to vGPU type
numDisplayHeads
Pointer to number of display heads

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuTypeId is invalid, or numDisplayHeads is NULL

- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve count of vGPU's supported display heads.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuTypeGetResolution (nvmlVgpuTypeId_t vgpuTypeId, unsigned int displayIndex, unsigned int *xdim, unsigned int *ydim)

Parameters

vgpuTypeId
Handle to vGPU type
displayIndex
Zero-based index of display head
xdim
Pointer to maximum number of pixels in X dimension
ydim
Pointer to maximum number of pixels in Y dimension

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuTypeId is invalid, or xdim or ydim are NULL, or displayIndex is out of range.
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve vGPU display head's maximum supported resolution.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuTypeGetLicense (nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeLicenseString, unsigned int size)

Parameters

vgpuTypeId
Handle to vGPU type
vgpuTypeLicenseString
Pointer to buffer to return license info
size
Size of vgpuTypeLicenseString buffer

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuTypeId is invalid, or vgpuTypeLicenseString is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if size is too small
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve license requirements for a vGPU type

The license type and version required to run the specified vGPU type is returned as an alphanumeric string, in the form "<license name>,<version>", for example "GRID-Virtual-PC,2.0". If a vGPU is runnable with* more than one type of license, the licenses are delimited by a semicolon, for example "GRID-Virtual-PC,2.0;GRID-Virtual-WS,2.0;GRID-Virtual-WS-Ext,2.0".

The total length of the returned string will not exceed 128 characters, including the NUL terminator. See nvmlVgpuConstants::NVML_GRID_LICENSE_BUFFER_SIZE.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuTypeGetFrameRateLimit (nvmlVgpuTypeId_t vgpuTypeId, unsigned int *frameRateLimit)

Parameters

vgpuTypeId
Handle to vGPU type
frameRateLimit
Reference to return the frame rate limit value

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_NOT_SUPPORTED if frame rate limiter is turned off for the vGPU type
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuTypeId is invalid, or frameRateLimit is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the static frame rate limit value of the vGPU type

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuTypeGetMaxInstances (nvmlDevice_t device, nvmlVgpuTypeId_t vgpuTypeId, unsigned int *vgpuInstanceCount)

Parameters

device
The identifier of the target device
vgpuTypeId
Handle to vGPU type
vgpuInstanceCount
Pointer to get the max number of vGPU instances that can be created on a deicve for given vgpuTypeId

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuTypeId is invalid or is not supported on target device, or vgpuInstanceCount is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the maximum number of vGPU instances creatable on a device for given vGPU type

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuTypeGetMaxInstancesPerVm (nvmlVgpuTypeId_t vgpuTypeId, unsigned int *vgpuInstanceCountPerVm)

Parameters

vgpuTypeId
Handle to vGPU type

vgpuInstanceCountPerVm
Pointer to get the max number of vGPU instances supported per VM for given vgpuTypeId

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuTypeId is invalid, or vgpuInstanceCountPerVm is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the maximum number of vGPU instances supported per VM for given vGPU type

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetActiveVgpus (nvmlDevice_t device, unsigned int *vgpuCount, nvmlVgpuInstance_t *vgpuInstances)

Parameters

device
The identifier of the target device

vgpuCount
Pointer which passes in the array size as well as get back the number of types

vgpuInstances
Pointer to array in which to return list of vGPU instances

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or vgpuCount is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if size is too small
- NVML_ERROR_NOT_SUPPORTED if vGPU is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the active vGPU instances on a device.

An array of active vGPU instances is returned in the caller-supplied buffer pointed at by vgpuInstances. The array element count is passed in vgpuCount, and vgpuCount is used to return the number of vGPU instances written to the buffer.

If the supplied buffer is not large enough to accommodate the vGPU instance array, the function returns NVML_ERROR_INSUFFICIENT_SIZE, with the element count of nvmlVgpuInstance_t array required in vgpuCount. To query the number of active vGPU instances, call this function with *vgpuCount = 0. The code will return NVML_ERROR_INSUFFICIENT_SIZE, or NVML_SUCCESS if no vGPU Types are supported.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetVmID (nvmlVgpuInstance_t vgpuInstance, char *vmId, unsigned int size, nvmlVgpuVmIdType_t *vmIdType)

Parameters

vgpuInstance
Identifier of the target vGPU instance

vmId
Pointer to caller-supplied buffer to hold VM ID

size
Size of buffer in bytes

vmIdType
Pointer to hold VM ID type

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vmId or vmIdType is NULL, or vgpuInstance is 0
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_INSUFFICIENT_SIZE if size is too small
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the VM ID associated with a vGPU instance.

The VM ID is returned as a string, not exceeding 80 characters in length (including the NUL terminator). See nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE.

The format of the VM ID varies by platform, and is indicated by the type identifier returned in vmIdType.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetUUID (nvmlVgpuInstance_t vgpuInstance, char *uuid, unsigned int size)

Parameters

vgpuInstance
Identifier of the target vGPU instance
uuid
Pointer to caller-supplied buffer to hold vGPU UUID
size
Size of buffer in bytes

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or uuid is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_INSUFFICIENT_SIZE if size is too small
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the UUID of a vGPU instance.

The UUID is a globally unique identifier associated with the vGPU, and is returned as a 5-part hexadecimal string, not exceeding 80 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetVmDriverVersion (nvmlVgpuInstance_t vgpuInstance, char *version, unsigned int length)

Parameters

vgpuInstance
Identifier of the target vGPU instance

version
Caller-supplied buffer to return driver version string
length
Size of version buffer

Returns

- NVML_SUCCESS if version has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the NVIDIA driver version installed in the VM associated with a vGPU.

The version is returned as an alphanumeric string in the caller-supplied buffer version. The length of the version string will not exceed 80 characters in length (including the NUL terminator). See nvmlConstants::NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE.

nvmlVgpuInstanceGetVmDriverVersion() may be called at any time for a vGPU instance. The guest VM driver version is returned as "Not Available" if no NVIDIA driver is installed in the VM, or the VM has not yet booted to the point where the NVIDIA driver is loaded and initialized.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetFbUsage (nvmlVgpuInstance_t vgpuInstance, unsigned long long *fbUsage)

Parameters

vgpuInstance
The identifier of the target instance
fbUsage
Pointer to framebuffer usage in bytes

Returns

- NVML_SUCCESS successful completion

- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or fbUsage is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the framebuffer usage in bytes.

Framebuffer usage is the amont of vGPU framebuffer memory that is currently in use by the VM.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetLicenseStatus (nvmlVgpuInstance_t vgpuInstance, unsigned int *licensed)

Parameters

vgpuInstance
Identifier of the target vGPU instance

licensed
Reference to return the licensing status

Returns

- NVML_SUCCESS if licensed has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or licensed is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Deprecated Use nvmlVgpuInstanceGetLicenseInfo_v2.

Retrieve the current licensing state of the vGPU instance.

If the vGPU is currently licensed, licensed is set to 1, otherwise it is set to 0.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetType (nvmlVgpuInstance_t vgpuInstance, nvmlVgpuTypeId_t *vgpuTypeId)

Parameters

vgpuInstance
Identifier of the target vGPU instance

vgpuTypeId
Reference to return the vgpuTypeId

Returns

- NVML_SUCCESS if vgpuTypeId has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or vgpuTypeId is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the vGPU type of a vGPU instance.

Returns the vGPU type ID of vgpu assigned to the vGPU instance.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetFrameRateLimit (nvmlVgpuInstance_t vgpuInstance, unsigned int *frameRateLimit)

Parameters

vgpuInstance
Identifier of the target vGPU instance

frameRateLimit
Reference to return the frame rate limit

Returns

- NVML_SUCCESS if frameRateLimit has been set

- NVML_ERROR_NOT_SUPPORTED if frame rate limiter is turned off for the vGPU type
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or frameRateLimit is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the frame rate limit set for the vGPU instance.

Returns the value of the frame rate limit set for the vGPU instance

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetEccMode (nvmlVgpuInstance_t vgpuInstance, nvmlEnableState_t *eccMode)

Parameters

vgpuInstance
The identifier of the target vGPU instance

eccMode
Reference in which to return the current ECC mode

Returns

- NVML_SUCCESS if the vgpuInstance's ECC mode has been successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or mode is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_NOT_SUPPORTED if the vGPU doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the current ECC mode of vGPU instance.

nvmlReturn_t nvmlVgpuInstanceGetEncoderCapacity (nvmlVgpuInstance_t vgpuInstance, unsigned int *encoderCapacity)

Parameters

vgpuInstance
Identifier of the target vGPU instance
encoderCapacity
Reference to an unsigned int for the encoder capacity

Returns

- NVML_SUCCESS if encoderCapacity has been retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or encoderQueryType is invalid
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the encoder capacity of a vGPU instance, as a percentage of maximum encoder capacity with valid values in the range 0-100.

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceSetEncoderCapacity (nvmlVgpuInstance_t vgpuInstance, unsigned int encoderCapacity)

Parameters

vgpuInstance
Identifier of the target vGPU instance
encoderCapacity
Unsigned int for the encoder capacity value

Returns

- NVML_SUCCESS if encoderCapacity has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or encoderCapacity is out of range of 0-100.
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Set the encoder capacity of a vGPU instance, as a percentage of maximum encoder capacity with valid values in the range 0-100.

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetEncoderStats (nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, unsigned int *averageFps, unsigned int *averageLatency)

Parameters

vgpuInstance
Identifier of the target vGPU instance
sessionCount
Reference to an unsigned int for count of active encoder sessions
averageFps
Reference to an unsigned int for trailing average FPS of all active sessions
averageLatency
Reference to an unsigned int for encode latency in microseconds

Returns

- NVML_SUCCESS if sessionCount, averageFps and averageLatency is fetched
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if sessionCount , or averageFps or averageLatency is NULL or vgpuInstance is 0.
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the current encoder statistics of a vGPU Instance

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetEncoderSessions (nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, nvmlEncoderSessionInfo_t *sessionInfo)

Parameters

vgpuInstance
Identifier of the target vGPU instance

sessionCount
Reference to caller supplied array size, and returns the number of sessions.

sessionInfo
Reference to caller supplied array in which the list of session information us returned.

Returns

- NVML_SUCCESS if sessionInfo is fetched
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INSUFFICIENT_SIZE if sessionCount is too small, array element count is returned in sessionCount
- NVML_ERROR_INVALID_ARGUMENT if sessionCount is NULL, or vgpuInstance is 0.
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves information about all active encoder sessions on a vGPU Instance.

An array of active encoder sessions is returned in the caller-supplied buffer pointed at by sessionInfo. The array element count is passed in sessionCount, and sessionCount is used to return the number of sessions written to the buffer.

If the supplied buffer is not large enough to accommodate the active session array, the function returns NVML_ERROR_INSUFFICIENT_SIZE, with the element count of nvmlEncoderSessionInfo_t array required in sessionCount. To query the number of active encoder sessions, call this function with *sessionCount = 0. The code will return NVML_SUCCESS with number of active encoder sessions updated in *sessionCount.

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetFBCStats (nvmlVgpuInstance_t vgpuInstance, nvmlFBCStats_t *fbcStats)

Parameters

vgpuInstance
Identifier of the target vGPU instance

fbcStats
Reference to nvmlFBCStats_t structure containing NvFBC stats

Returns

- NVML_SUCCESS if fbcStats is fetched
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or fbcStats is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the active frame buffer capture sessions statistics of a vGPU Instance

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetFBCSessions (nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, nvmlFBCSessionInfo_t *sessionInfo)

Parameters

vgpuInstance
Identifier of the target vGPU instance

sessionCount
Reference to caller supplied array size, and returns the number of sessions.

sessionInfo
Reference in which to return the session information

Returns

- NVML_SUCCESS if sessionInfo is fetched
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or sessionCount is NULL.
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_INSUFFICIENT_SIZE if sessionCount is too small, array element count is returned in sessionCount
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves information about active frame buffer capture sessions on a vGPU Instance.

An array of active FBC sessions is returned in the caller-supplied buffer pointed at by sessionInfo. The array element count is passed in sessionCount, and sessionCount is used to return the number of sessions written to the buffer.

If the supplied buffer is not large enough to accommodate the active session array, the function returns NVML_ERROR_INSUFFICIENT_SIZE, with the element count of nvmlFBCSessionInfo_t array required in sessionCount. To query the number of active FBC sessions, call this function with *sessionCount = 0. The code will return NVML_SUCCESS with number of active FBC sessions updated in *sessionCount.

For Maxwell or newer fully supported devices.

hResolution, vResolution, averageFPS and averageLatency data for a FBC session returned in sessionInfo may be zero if there are no new frames captured since the session started.

nvmlReturn_t nvmlVgpuInstanceGetGpuInstanceId (nvmlVgpuInstance_t vgpuInstance, unsigned int *gpuInstanceId)

Parameters

vgpuInstance
Identifier of the target vGPU instance

gpuInstanceId
GPU Instance ID

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or gpuInstanceId is NULL.
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the GPU Instance ID for the given vGPU Instance. The API will return a valid GPU Instance ID for MIG backed vGPU Instance, else INVALID_GPU_INSTANCE_ID is returned.

For Kepler or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetGpuPciId (nvmlVgpuInstance_t vgpuInstance, char *vgpuPciId, unsigned int *length)

Parameters

vgpuInstance
Identifier of the target vGPU instance

vgpuPciId
Caller-supplied buffer to return vGPU PCI Id string

length
Size of the vgpuPciId buffer

Returns

- NVML_SUCCESS if vGPU PCI Id is sucessfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or vgpuPciId is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_DRIVER_NOT_LOADED if NVIDIA driver is not running on the vGPU instance
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small, length is set to required length
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the PCI Id of the given vGPU Instance i.e. the PCI Id of the GPU as seen inside the VM.

The vGPU PCI id is returned as "00000000:00:00.0" if NVIDIA driver is not installed on the vGPU instance.

nvmlReturn_t nvmlVgpuTypeGetCapabilities (nvmlVgpuTypeId_t vgpuTypeId, nvmlVgpuCapability_t capability, unsigned int *capResult)

Parameters

vgpuTypeId
Handle to vGPU type

capability
Specifies the nvmlVgpuCapability_t to be queried

capResult
A boolean for the queried capability indicating that feature is supported

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuTypeId is invalid, or capability is invalid, or capResult is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the requested capability for a given vGPU type. Refer to the nvmlVgpuCapability_t structure for the specific capabilities that can be queried. The return value in capResult should be treated as a boolean, with a non-zero value indicating that the capability is supported.

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetMdevUUID (nvmlVgpuInstance_t vgpuInstance, char *mdevUuid, unsigned int size)

Parameters

vgpuInstance
Identifier of the target vGPU instance

mdevUuid
Pointer to caller-supplied buffer to hold MDEV UUID
size
Size of buffer in bytes

Returns

- NVML_SUCCESS successful completion
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_SUPPORTED on any hypervisor other than KVM
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or mdevUuid is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_INSUFFICIENT_SIZE if size is too small
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the MDEV UUID of a vGPU instance.

The MDEV UUID is a globally unique identifier of the mdev device assigned to the VM, and is returned as a 5-part hexadecimal string, not exceeding 80 characters in length (including the NULL terminator). MDEV UUID is displayed only on KVM platform. See nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE.

For Maxwell or newer fully supported devices.

4.25. vGPU Migration

This chapter describes operations that are associated with vGPU Migration.

struct nvmlVgpuVersion_t

struct nvmlVgpuMetadata_t

struct nvmlVgpuPgpuMetadata_t

struct nvmlVgpuPgpuCompatibility_t

enum nvmlVgpuVmCompatibility_t

vGPU VM compatibility codes

Values

NVML_VGPU_VM_COMPATIBILITY_NONE = 0x0
vGPU is not runnable
NVML_VGPU_VM_COMPATIBILITY_COLD = 0x1
vGPU is runnable from a cold / powered-off state (ACPI S5)
NVML_VGPU_VM_COMPATIBILITY_HIBERNATE = 0x2
vGPU is runnable from a hibernated state (ACPI S4)
NVML_VGPU_VM_COMPATIBILITY_SLEEP = 0x4
vGPU is runnable from a sleeped state (ACPI S3)
NVML_VGPU_VM_COMPATIBILITY_LIVE = 0x8
vGPU is runnable from a live/paused (ACPI S0)

enum nvmlVgpuPgpuCompatibilityLimitCode_t

vGPU-pGPU compatibility limit codes

Values

NVML_VGPU_COMPATIBILITY_LIMIT_NONE = 0x0
Compatibility is not limited.
NVML_VGPU_COMPATIBILITY_LIMIT_HOST_DRIVER = 0x1
ompatibility is limited by host driver version.
NVML_VGPU_COMPATIBILITY_LIMIT_GUEST_DRIVER = 0x2
Compatibility is limited by guest driver version.
NVML_VGPU_COMPATIBILITY_LIMIT_GPU = 0x4
Compatibility is limited by GPU hardware.
NVML_VGPU_COMPATIBILITY_LIMIT_OTHER = 0x80000000
Compatibility is limited by an undefined factor.

nvmlReturn_t nvmlVgpuInstanceGetMetadata (nvmlVgpuInstance_t vgpuInstance, nvmlVgpuMetadata_t *vgpuMetadata, unsigned int *bufferSize)

Parameters

vgpuInstance
vGPU instance handle

vgpuMetadata
Pointer to caller-supplied buffer into which vGPU metadata is written

bufferSize
Size of vgpuMetadata buffer

Returns

- NVML_SUCCESS vGPU metadata structure was successfully returned
- NVML_ERROR_INSUFFICIENT_SIZE vgpuMetadata buffer is too small, required size is returned in bufferSize
- NVML_ERROR_INVALID_ARGUMENT if bufferSize is NULL or vgpuInstance is 0; if vgpuMetadata is NULL and the value of bufferSize is not 0.
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Returns vGPU metadata structure for a running vGPU. The structure contains information about the vGPU and its associated VM such as the currently installed NVIDIA guest driver version, together with host driver version and an opaque data section containing internal state.

nvmlVgpuInstanceGetMetadata() may be called at any time for a vGPU instance. Some fields in the returned structure are dependent on information obtained from the guest VM, which may not yet have reached a state where that information is available. The current state of these dependent fields is reflected in the info structure's nvmlVgpuGuestInfoState_t field.

The VMM may choose to read and save the vGPU's VM info as persistent metadata associated with the VM, and provide it to Virtual GPU Manager when creating a vGPU for subsequent instances of the VM.

The caller passes in a buffer via vgpuMetadata, with the size of the buffer in bufferSize. If the vGPU Metadata structure is too large to fit in the supplied buffer, the function returns NVML_ERROR_INSUFFICIENT_SIZE with the size needed in bufferSize.

nvmlReturn_t nvmlDeviceGetVgpuMetadata (nvmlDevice_t device, nvmlVgpuPgpuMetadata_t *pgpuMetadata, unsigned int *bufferSize)

Parameters

device
The identifier of the target device

pgpuMetadata
Pointer to caller-supplied buffer into which pgpuMetadata is written

bufferSize
Pointer to size of pgpuMetadata buffer

Returns

- NVML_SUCCESS GPU metadata structure was successfully returned
- NVML_ERROR_INSUFFICIENT_SIZE pgpuMetadata buffer is too small, required size is returned in bufferSize
- NVML_ERROR_INVALID_ARGUMENT if bufferSize is NULL or device is invalid; if pgpuMetadata is NULL and the value of bufferSize is not 0.
- NVML_ERROR_NOT_SUPPORTED vGPU is not supported by the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Returns a vGPU metadata structure for the physical GPU indicated by device. The structure contains information about the GPU and the currently installed NVIDIA host driver version that's controlling it, together with an opaque data section containing internal state.

The caller passes in a buffer via pgpuMetadata, with the size of the buffer in bufferSize. If the pgpuMetadata structure is too large to fit in the supplied buffer, the function returns NVML_ERROR_INSUFFICIENT_SIZE with the size needed in bufferSize.

nvmlReturn_t nvmlGetVgpuCompatibility (nvmlVgpuMetadata_t *vgpuMetadata,

nvmlVgpuPgpuMetadata_t *pgpuMetadata, nvmlVgpuPgpuCompatibility_t *compatibilityInfo)

Parameters

vgpuMetadata
Pointer to caller-supplied vGPU metadata structure

pgpuMetadata
Pointer to caller-supplied GPU metadata structure

compatibilityInfo
Pointer to caller-supplied buffer to hold compatibility info

Returns

- NVML_SUCCESS vGPU metadata structure was successfully returned
- NVML_ERROR_INVALID_ARGUMENT if vgpuMetadata or pgpuMetadata or bufferSize are NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Takes a vGPU instance metadata structure read from nvmlVgpuInstanceGetMetadata(), and a vGPU metadata structure for a physical GPU read from nvmlDeviceGetVgpuMetadata(), and returns compatibility information of the vGPU instance and the physical GPU.

The caller passes in a buffer via compatibilityInfo, into which a compatibility information structure is written. The structure defines the states in which the vGPU / VM may be booted on the physical GPU. If the vGPU / VM compatibility with the physical GPU is limited, a limit code indicates the factor limiting compatability. (see nvmlVgpuPgpuCompatibilityLimitCode_t for details).

Note: vGPU compatibility does not take into account dynamic capacity conditions that may limit a system's ability to boot a given vGPU or associated VM.

nvmlReturn_t nvmlDeviceGetPgpuMetadataString (nvmlDevice_t device, char *pgpuMetadata, unsigned int *bufferSize)

Parameters

device
The identifier of the target device

pgpuMetadata
Pointer to caller-supplied buffer into which pgpuMetadata is written
bufferSize
Pointer to size of pgpuMetadata buffer

Returns

- NVML_SUCCESS GPU metadata structure was successfully returned
- NVML_ERROR_INSUFFICIENT_SIZE pgpuMetadata buffer is too small, required size is returned in bufferSize
- NVML_ERROR_INVALID_ARGUMENT if bufferSize is NULL or device is invalid; if pgpuMetadata is NULL and the value of bufferSize is not 0.
- NVML_ERROR_NOT_SUPPORTED if vGPU is not supported by the system
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Returns the properties of the physical GPU indicated by the device in an ascii-encoded string format.

The caller passes in a buffer via pgpuMetadata, with the size of the buffer in bufferSize. If the string is too large to fit in the supplied buffer, the function returns NVML_ERROR_INSUFFICIENT_SIZE with the size needed in bufferSize.

nvmlReturn_t nvmlDeviceGetVgpuSchedulerLog (nvmlDevice_t device, nvmlVgpuSchedulerLog_t *pSchedulerLog)

Parameters

device
The identifier of the target device
pSchedulerLog
Reference in which pSchedulerLog is written

Returns

- NVML_SUCCESS vGPU scheduler logs were successfully obtained
- NVML_ERROR_INVALID_ARGUMENT if pSchedulerLog is NULL or device is invalid
- NVML_ERROR_NOT_SUPPORTED The API is not supported in current state or device not in vGPU host mode
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Returns the vGPU Software scheduler logs. pSchedulerLog points to a caller-allocated structure to contain the logs. The number of elements returned will never exceed NVML_SCHEDULER_SW_MAX_LOG_ENTRIES.

To get the entire logs, call the function atleast 5 times a second.

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetVgpuSchedulerState (nvmlDevice_t device, nvmlVgpuSchedulerGetState_t *pSchedulerState)

Parameters

device
The identifier of the target device

pSchedulerState
Reference in which pSchedulerState is returned

Returns

- NVML_SUCCESS vGPU scheduler state is successfully obtained
- NVML_ERROR_INVALID_ARGUMENT if pSchedulerState is NULL or device is invalid
- NVML_ERROR_NOT_SUPPORTED The API is not supported in current state or device not in vGPU host mode
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Returns the vGPU scheduler state. The information returned in nvmlVgpuSchedulerGetState_t is not relevant if the BEST EFFORT policy is set.

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceGetVgpuSchedulerCapabilities (nvmlDevice_t device, nvmlVgpuSchedulerCapabilities_t *pCapabilities)

Parameters

device
The identifier of the target device
pCapabilities
Reference in which pCapabilities is written

Returns

- NVML_SUCCESS vGPU scheduler capabilities were successfully obtained
- NVML_ERROR_INVALID_ARGUMENT if pCapabilities is NULL or device is invalid
- NVML_ERROR_NOT_SUPPORTED The API is not supported in current state or device not in vGPU host mode
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Returns the vGPU scheduler capabilities. The list of supported vGPU schedulers returned in nvmlVgpuSchedulerCapabilities_t is from the NVML_VGPU_SCHEDULER_POLICY_*. This list enumerates the supported scheduler policies if the engine is Graphics type. The other values in nvmlVgpuSchedulerCapabilities_t are also applicable if the engine is Graphics type. For other engine types, it is BEST EFFORT policy. If ARR is supported and enabled, scheduling frequency and averaging factor are applicable else timeSlice is applicable.

For Pascal or newer fully supported devices.

nvmlReturn_t nvmlDeviceSetVgpuSchedulerState (nvmlDevice_t device, nvmlVgpuSchedulerSetState_t *pSchedulerState)

Parameters

device
The identifier of the target device
pSchedulerState
vGPU pSchedulerState to set

Returns

- NVML_SUCCESS vGPU scheduler state has been successfully set
- NVML_ERROR_INVALID_ARGUMENT if pSchedulerState is NULL or device is invalid
- NVML_ERROR_RESET_REQUIRED if setting pSchedulerState failed with fatal error, reboot is required to overcome from this error.
- NVML_ERROR_NOT_SUPPORTED The API is not supported in current state or device not in vGPU host mode or if any vGPU instance currently exists on the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Sets the vGPU scheduler state.

For Pascal or newer fully supported devices.

The scheduler state change won't persist across module load/unload. Scheduler state and params will be allowed to set only when no VM is running. In nvmlVgpuSchedulerSetState_t, IFF enableARRMode is enabled then provide avgFactorForARR and frequency as input. If enableARRMode is disabled then provide timeslice as input.

nvmlReturn_t nvmlGetVgpuVersion (nvmlVgpuVersion_t *supported, nvmlVgpuVersion_t *current)

Parameters

supported
Pointer to the structure in which the preset range of vGPU versions supported by the NVIDIA vGPU Manager is written

current
Pointer to the structure in which the range of supported vGPU versions set by an administrator is written

Returns

- NVML_SUCCESS The vGPU version range structures were successfully obtained.
- NVML_ERROR_NOT_SUPPORTED The API is not supported.
- NVML_ERROR_INVALID_ARGUMENT The supported parameter or the current parameter is NULL.
- NVML_ERROR_UNKNOWN An error occurred while the data was being fetched.

Description

Query the ranges of supported vGPU versions.

This function gets the linear range of supported vGPU versions that is preset for the NVIDIA vGPU Manager and the range set by an administrator. If the preset range has not been overridden by nvmlSetVgpuVersion, both ranges are the same.

The caller passes pointers to the following nvmlVgpuVersion_t structures, into which the NVIDIA vGPU Manager writes the ranges: 1. supported structure that represents the preset range of vGPU versions supported by the NVIDIA vGPU Manager. 2. current structure that represents the range of supported vGPU versions set by an administrator. By default, this range is the same as the preset range.

nvmlReturn_t nvmlSetVgpuVersion (nvmlVgpuVersion_t *vgpuVersion)

Parameters

vgpuVersion
Pointer to a caller-supplied range of supported vGPU versions.

Returns

- NVML_SUCCESS The preset range of supported vGPU versions was successfully overridden.
- NVML_ERROR_NOT_SUPPORTED The API is not supported.
- NVML_ERROR_IN_USE The range was not overridden because a VM is running on the host.
- NVML_ERROR_INVALID_ARGUMENT The vgpuVersion parameter specifies a range that is outside the range supported by the NVIDIA vGPU Manager or if vgpuVersion is NULL.

Description

Override the preset range of vGPU versions supported by the NVIDIA vGPU Manager with a range set by an administrator.

This function configures the NVIDIA vGPU Manager with a range of supported vGPU versions set by an administrator. This range must be a subset of the preset range that the NVIDIA vGPU Manager supports. The custom range set by an administrator takes precedence over the preset range and is advertised to the guest VM for negotiating the vGPU version. See nvmlGetVgpuVersion for details of how to query the preset range of versions supported.

This function takes a pointer to vGPU version range structure nvmlVgpuVersion_t as input to override the preset vGPU version range that the NVIDIA vGPU Manager supports.

After host system reboot or driver reload, the range of supported versions reverts to the range that is preset for the NVIDIA vGPU Manager.

1. The range set by the administrator must be a subset of the preset range that the NVIDIA vGPU Manager supports. Otherwise, an error is returned. 2. If the range of supported guest driver versions does not overlap the range set by the administrator, the guest driver fails to load. 3. If the range of supported guest driver versions overlaps the range set by the administrator, the guest driver will load with a negotiated vGPU version that is the maximum value in the overlapping range. 4. No VMs must be running on the host when this function is called. If a VM is running on the host, the call to this function fails.

4.26. vGPU Utilization and Accounting

This chapter describes operations that are associated with vGPU Utilization and Accounting.

nvmlReturn_t nvmlDeviceGetVgpuUtilization (nvmlDevice_t device, unsigned long long lastSeenTimeStamp, nvmlValueType_t *sampleValType, unsigned int *vgpuInstanceSamplesCount, nvmlVgpuInstanceUtilizationSample_t *utilizationSamples)

Parameters

device
The identifier for the target device

lastSeenTimeStamp
Return only samples with timestamp greater than lastSeenTimeStamp.

sampleValType
Pointer to caller-supplied buffer to hold the type of returned sample values

vgpuInstanceSamplesCount
Pointer to caller-supplied array size, and returns number of vGPU instances

utilizationSamples
Pointer to caller-supplied buffer in which vGPU utilization samples are returned

Returns

- NVML_SUCCESS if utilization samples are successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, vgpuInstanceSamplesCount or sampleValType is NULL, or a sample count of 0 is passed with a non-NULL utilizationSamples
- NVML_ERROR_INSUFFICIENT_SIZE if supplied vgpuInstanceSamplesCount is too small to return samples for all vGPU instances currently executing on the device
- NVML_ERROR_NOT_SUPPORTED if vGPU is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_NOT_FOUND if sample entries are not found
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves current utilization for vGPUs on a physical GPU (device).

For Kepler or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, and video decoder for vGPU instances running on a device. Utilization values are returned as an array of utilization sample structures in the caller-supplied buffer pointed at by utilizationSamples. One utilization sample structure is returned per vGPU instance, and includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "unsigned int" values in nvmlValue_t unions. The function sets the caller-supplied sampleValType to NVML_VALUE_TYPE_UNSIGNED_INT to indicate the returned value type.

To read utilization values, first determine the size of buffer required to hold the samples by invoking the function with utilizationSamples set to NULL. The function will return NVML_ERROR_INSUFFICIENT_SIZE, with the current vGPU instance count in vgpuInstanceSamplesCount, or NVML_SUCCESS if the current vGPU instance count is zero. The caller should allocate a buffer of size vgpuInstanceSamplesCount * sizeof(nvmlVgpuInstanceUtilizationSample_t). Invoke the function again with the allocated buffer passed in utilizationSamples, and vgpuInstanceSamplesCount set to the number of entries the buffer is sized for.

On successful return, the function updates vgpuInstanceSampleCount with the number of vGPU utilization sample structures that were actually written. This may differ from a previously read value as vGPU instances are created or destroyed.

lastSeenTimeStamp represents the CPU timestamp in microseconds at which utilization samples were last read. Set it to 0 to read utilization based on all the samples maintained by the driver's internal sample buffer. Set lastSeenTimeStamp to a timeStamp retrieved from a previous query to read utilization since the previous query.

nvmlReturn_t nvmlDeviceGetVgpuInstancesUtilizationInfo (nvmlDevice_t device, nvmlVgpuInstancesUtilizationInfo_t *vgpuUtilInfo)

Parameters

device
The identifier for the target device

vgpuUtilInfo
Pointer to the caller-provided structure of nvmlVgpuInstancesUtilizationInfo_t

Returns

- NVML_SUCCESS if utilization samples are successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, vgpuUtilInfo is NULL, or vgpuUtilInfo->vgpuInstanceCount is 0
- NVML_ERROR_NOT_SUPPORTED if vGPU is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_VERSION_MISMATCH if the version of vgpuUtilInfo is invalid
- NVML_ERROR_INSUFFICIENT_SIZE if vgpuUtilInfo->vgpuUtilArray is NULL, or the buffer size of vgpuUtilInfo->vgpuInstanceCount is too small. The caller should check the current vGPU instance count from the returned vgpuUtilInfo->vgpuInstanceCount, and call the function again with a buffer of size vgpuUtilInfo->vgpuInstanceCount * sizeof(nvmlVgpuInstanceUtilizationInfo_t)
- NVML_ERROR_NOT_FOUND if sample entries are not found
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves recent utilization for vGPU instances running on a physical GPU (device).

For Kepler or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, video decoder, jpeg decoder, and OFA for vGPU instances running on a device. Utilization values are returned as an array of utilization sample structures in the caller-supplied buffer pointed at by vgpuUtilInfo->vgpuUtilArray. One utilization sample structure is returned per vGPU instance, and includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "unsigned int" values in

nvmlValue_t unions. The function sets the caller-supplied vgpuUtilInfo->sampleValType to NVML_VALUE_TYPE_UNSIGNED_INT to indicate the returned value type.

To read utilization values, first determine the size of buffer required to hold the samples by invoking the function with vgpuUtilInfo->vgpuUtilArray set to NULL. The function will return NVML_ERROR_INSUFFICIENT_SIZE, with the current vGPU instance count in vgpuUtilInfo->vgpuInstanceCount, or NVML_SUCCESS if the current vGPU instance count is zero. The caller should allocate a buffer of size vgpuUtilInfo->vgpuInstanceCount * sizeof(nvmlVgpuInstanceUtilizationInfo_t). Invoke the function again with the allocated buffer passed in vgpuUtilInfo->vgpuUtilArray, and vgpuUtilInfo->vgpuInstanceCount set to the number of entries the buffer is sized for.

On successful return, the function updates vgpuUtilInfo->vgpuInstanceCount with the number of vGPU utilization sample structures that were actually written. This may differ from a previously read value as vGPU instances are created or destroyed.

vgpuUtilInfo->lastSeenTimeStamp represents the CPU timestamp in microseconds at which utilization samples were last read. Set it to 0 to read utilization based on all the samples maintained by the driver's internal sample buffer. Set vgpuUtilInfo->lastSeenTimeStamp to a timeStamp retrieved from a previous query to read utilization since the previous query.

nvmlReturn_t nvmlDeviceGetVgpuProcessUtilization (nvmlDevice_t device, unsigned long long lastSeenTimeStamp, unsigned int *vgpuProcessSamplesCount, nvmlVgpuProcessUtilizationSample_t *utilizationSamples)

Parameters

device
The identifier for the target device

lastSeenTimeStamp
Return only samples with timestamp greater than lastSeenTimeStamp.

vgpuProcessSamplesCount
Pointer to caller-supplied array size, and returns number of processes running on vGPU instances

utilizationSamples
Pointer to caller-supplied buffer in which vGPU sub process utilization samples are returned

Returns

- NVML_SUCCESS if utilization samples are successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, vgpuProcessSamplesCount or a sample count of 0 is passed with a non-NULL utilizationSamples
- NVML_ERROR_INSUFFICIENT_SIZE if supplied vgpuProcessSamplesCount is too small to return samples for all vGPU instances currently executing on the device
- NVML_ERROR_NOT_SUPPORTED if vGPU is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_NOT_FOUND if sample entries are not found
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves current utilization for processes running on vGPUs on a physical GPU (device).

For Maxwell or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, and video decoder for processes running on vGPU instances active on a device. Utilization values are returned as an array of utilization sample structures in the caller-supplied buffer pointed at by utilizationSamples. One utilization sample structure is returned per process running on vGPU instances, that had some non-zero utilization during the last sample period. It includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "unsigned int" values.

To read utilization values, first determine the size of buffer required to hold the samples by invoking the function with utilizationSamples set to NULL. The function will return NVML_ERROR_INSUFFICIENT_SIZE, with the current vGPU instance count in vgpuProcessSamplesCount. The caller should allocate a buffer of size vgpuProcessSamplesCount * sizeof(nvmlVgpuProcessUtilizationSample_t). Invoke the function again with the allocated buffer passed in utilizationSamples, and vgpuProcessSamplesCount set to the number of entries the buffer is sized for.

On successful return, the function updates vgpuSubProcessSampleCount with the number of vGPU sub process utilization sample structures that were actually written. This may differ from a previously read value depending on the number of processes that are active in any given sample period.

lastSeenTimeStamp represents the CPU timestamp in microseconds at which utilization samples were last read. Set it to 0 to read utilization based on all the samples maintained by the driver's internal sample buffer. Set lastSeenTimeStamp to a timeStamp retrieved from a previous query to read utilization since the previous query.

nvmlReturn_t nvmlDeviceGetVgpuProcessesUtilizationInfo (nvmlDevice_t device, nvmlVgpuProcessesUtilizationInfo_t *vgpuProcUtilInfo)

Parameters

device
The identifier for the target device

vgpuProcUtilInfo
Pointer to the caller-provided structure of nvmlVgpuProcessesUtilizationInfo_t

Returns

- NVML_SUCCESS if utilization samples are successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or vgpuProcUtilInfo is null
- NVML_ERROR_VERSION_MISMATCH if the version of vgpuProcUtilInfo is invalid
- NVML_ERROR_INSUFFICIENT_SIZE if vgpuProcUtilInfo->vgpuProcUtilArray is null, or supplied vgpuProcUtilInfo->vgpuProcessCount is too small to return samples for all processes on vGPU instances currently executing on the device. The caller should check the current processes count from the returned vgpuProcUtilInfo->vgpuProcessCount, and call the function again with a buffer of size vgpuProcUtilInfo->vgpuProcessCount * sizeof(nvmlVgpuProcessUtilizationSample_t)
- NVML_ERROR_NOT_SUPPORTED if vGPU is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_NOT_FOUND if sample entries are not found
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves recent utilization for processes running on vGPU instances on a physical GPU (device).

For Maxwell or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, video decoder, jpeg decoder, and OFA for processes running on vGPU instances active on a device. Utilization values are returned as an array of utilization sample structures in

the caller-supplied buffer pointed at by vgpuProcUtilInfo->vgpuProcUtilArray. One utilization sample structure is returned per process running on vGPU instances, that had some non-zero utilization during the last sample period. It includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "unsigned int" values.

To read utilization values, first determine the size of buffer required to hold the samples by invoking the function with vgpuProcUtilInfo->vgpuProcUtilArray set to NULL. The function will return NVML_ERROR_INSUFFICIENT_SIZE, with the current processes' count running on vGPU instances in vgpuProcUtilInfo->vgpuProcessCount. The caller should allocate a buffer of size vgpuProcUtilInfo->vgpuProcessCount * sizeof(nvmlVgpuProcessUtilizationSample_t). Invoke the function again with the allocated buffer passed in vgpuProcUtilInfo->vgpuProcUtilArray, and vgpuProcUtilInfo->vgpuProcessCount set to the number of entries the buffer is sized for.

On successful return, the function updates vgpuProcUtilInfo->vgpuProcessCount with the number of vGPU sub process utilization sample structures that were actually written. This may differ from a previously read value depending on the number of processes that are active in any given sample period.

vgpuProcUtilInfo->lastSeenTimeStamp represents the CPU timestamp in microseconds at which utilization samples were last read. Set it to 0 to read utilization based on all the samples maintained by the driver's internal sample buffer. Set vgpuProcUtilInfo->lastSeenTimeStamp to a timeStamp retrieved from a previous query to read utilization since the previous query.

nvmlReturn_t nvmlVgpuInstanceGetAccountingMode (nvmlVgpuInstance_t vgpuInstance, nvmlEnableState_t *mode)

Parameters

vgpuInstance
The identifier of the target vGPU instance

mode
Reference in which to return the current accounting mode

Returns

- NVML_SUCCESS if the mode has been successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or mode is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system

- NVML_ERROR_NOT_SUPPORTED if the vGPU doesn't support this feature
- NVML_ERROR_DRIVER_NOT_LOADED if NVIDIA driver is not running on the vGPU instance
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Queries the state of per process accounting mode on vGPU.

For Maxwell or newer fully supported devices.

nvmlReturn_t nvmlVgpuInstanceGetAccountingPids (nvmlVgpuInstance_t vgpuInstance, unsigned int *count, unsigned int *pids)

Parameters

vgpuInstance
The identifier of the target vGPU instance

count
Reference in which to provide the pids array size, and to return the number of elements ready to be queried

pids
Reference in which to return list of process ids

Returns

- NVML_SUCCESS if pids were successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or count is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_NOT_SUPPORTED if the vGPU doesn't support this feature or accounting mode is disabled
- NVML_ERROR_INSUFFICIENT_SIZE if count is too small (count is set to expected value)
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Queries list of processes running on vGPU that can be queried for accounting stats. The list of processes returned can be in running or terminated state.

For Maxwell or newer fully supported devices.

To just query the maximum number of processes that can be queried, call this function with *count = 0 and pids=NULL. The return code will be NVML_ERROR_INSUFFICIENT_SIZE, or NVML_SUCCESS if list is empty.

For more details see nvmlVgpuInstanceGetAccountingStats.

In case of PID collision some processes might not be accessible before the circular buffer is full.

See also:

nvmlVgpuInstanceGetAccountingPids

nvmlReturn_t nvmlVgpuInstanceGetAccountingStats (nvmlVgpuInstance_t vgpuInstance, unsigned int pid, nvmlAccountingStats_t *stats)

Parameters

vgpuInstance
The identifier of the target vGPU instance

pid
Process Id of the target process to query stats for

stats
Reference in which to return the process's accounting stats

Returns

- NVML_SUCCESS if stats have been successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or stats is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system or stats is not found
- NVML_ERROR_NOT_SUPPORTED if the vGPU doesn't support this feature or accounting mode is disabled
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Queries process's accounting stats.

For Maxwell or newer fully supported devices.

Accounting stats capture GPU utilization and other statistics across the lifetime of a process, and can be queried during life time of the process or after its termination. The time field in nvmlAccountingStats_t is reported as 0 during the lifetime of the process and updated to actual running time after its termination. Accounting stats are kept in a circular buffer, newly created processes overwrite information about old processes.

See nvmlAccountingStats_t for description of each returned metric. List of processes that can be queried can be retrieved from nvmlVgpuInstanceGetAccountingPids.

- Accounting Mode needs to be on. See nvmlVgpuInstanceGetAccountingMode.
- Only compute and graphics applications stats can be queried. Monitoring applications stats can't be queried since they don't contribute to GPU utilization.
- In case of pid collision stats of only the latest process (that terminated last) will be reported

nvmlReturn_t nvmlVgpuInstanceClearAccountingPids (nvmlVgpuInstance_t vgpuInstance)

Parameters

vgpuInstance
The identifier of the target vGPU instance

Returns

- NVML_SUCCESS if accounting information has been cleared
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is invalid
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_NOT_SUPPORTED if the vGPU doesn't support this feature or accounting mode is disabled
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Clears accounting information of the vGPU instance that have already terminated.

For Maxwell or newer fully supported devices. Requires root/admin permissions.

- Accounting Mode needs to be on. See nvmlVgpuInstanceGetAccountingMode.
- Only compute and graphics applications stats are reported and can be cleared since monitoring applications stats don't contribute to GPU utilization.

nvmlReturn_t nvmlVgpuInstanceGetLicenseInfo_v2 (nvmlVgpuInstance_t vgpuInstance, nvmlVgpuLicenseInfo_t *licenseInfo)

Parameters

vgpuInstance
Identifier of the target vGPU instance

licenseInfo
Pointer to vGPU license information structure

Returns

- NVML_SUCCESS if information is successfully retrieved
- NVML_ERROR_INVALID_ARGUMENT if vgpuInstance is 0, or licenseInfo is NULL
- NVML_ERROR_NOT_FOUND if vgpuInstance does not match a valid active vGPU instance on the system
- NVML_ERROR_DRIVER_NOT_LOADED if NVIDIA driver is not running on the vGPU instance
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Query the license information of the vGPU instance.

For Maxwell or newer fully supported devices.

4.27. Excluded GPU Queries

This chapter describes NVML operations that are associated with excluded GPUs.

struct nvmlExcludedDeviceInfo_t

nvmlReturn_t nvmlGetExcludedDeviceCount (unsigned int *deviceCount)

Parameters

deviceCount
Reference in which to return the number of excluded devices

Returns

- NVML_SUCCESS if deviceCount has been set
- NVML_ERROR_INVALID_ARGUMENT if deviceCount is NULL

Description

Retrieves the number of excluded GPU devices in the system.

For all products.

nvmlReturn_t nvmlGetExcludedDeviceInfoByIndex (unsigned int index, nvmlExcludedDeviceInfo_t *info)

Parameters

index
The index of the target GPU, >= 0 and < deviceCount

info
Reference in which to return the device information

Returns

- NVML_SUCCESS if device has been set
- NVML_ERROR_INVALID_ARGUMENT if index is invalid or info is NULL

Description

Acquire the device information for an excluded GPU device, based on its index.

For all products.

Valid indices are derived from the deviceCount returned by nvmlGetExcludedDeviceCount(). For example, if deviceCount is 2 the valid indices are 0 and 1, corresponding to GPU 0 and GPU 1.

See also:

nvmlGetExcludedDeviceCount

4.28. Multi Instance GPU Management

This chapter describes NVML operations that are associated with Multi Instance GPU management.

struct nvmlGpuInstancePlacement_t

struct nvmlGpuInstanceProfileInfo_t

struct nvmlGpuInstanceProfileInfo_v2_t

struct nvmlGpuInstanceProfileInfo_v3_t

struct nvmlComputeInstanceProfileInfo_t

struct nvmlComputeInstanceProfileInfo_v2_t

struct nvmlComputeInstanceProfileInfo_v3_t

nvmlReturn_t nvmlDeviceSetMigMode (nvmlDevice_t device, unsigned int mode, nvmlReturn_t *activationStatus)

Parameters

device
The identifier of the target device

mode
The mode to be set, NVML_DEVICE_MIG_DISABLE or NVML_DEVICE_MIG_ENABLE

activationStatus
The activationStatus status

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device,mode or activationStatus are invalid
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation
- NVML_ERROR_NOT_SUPPORTED If device doesn't support MIG mode

Description

Set MIG mode for the device.

For Ampere or newer fully supported devices. Requires root user.

This mode determines whether a GPU instance can be created.

This API may unbind or reset the device to activate the requested mode. Thus, the attributes associated with the device, such as minor number, might change. The caller of this API is expected to query such attributes again.

On certain platforms like pass-through virtualization, where reset functionality may not be exposed directly, VM reboot is required. activationStatus would return NVML_ERROR_RESET_REQUIRED for such cases.

activationStatus would return the appropriate error code upon unsuccessful activation. For example, if device unbind fails because the device isn't idle, NVML_ERROR_IN_USE would be returned. The caller of this API is expected to idle the device and retry setting the mode.

On Windows, only disabling MIG mode is supported. activationStatus would return NVML_ERROR_NOT_SUPPORTED as GPU reset is not supported on Windows through this API.

nvmlReturn_t nvmlDeviceGetMigMode (nvmlDevice_t device, unsigned int *currentMode, unsigned int *pendingMode)

Parameters

device
The identifier of the target device

currentMode
Returns the current mode, NVML_DEVICE_MIG_DISABLE or NVML_DEVICE_MIG_ENABLE

pendingMode
Returns the pending mode, NVML_DEVICE_MIG_DISABLE or NVML_DEVICE_MIG_ENABLE

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized

- NVML_ERROR_INVALID_ARGUMENT If device, currentMode or pendingMode are invalid
- NVML_ERROR_NOT_SUPPORTED If device doesn't support MIG mode

Description

Get MIG mode for the device.

For Ampere or newer fully supported devices.

Changing MIG modes may require device unbind or reset. The "pending" MIG mode refers to the target mode following the next activation trigger.

nvmlReturn_t nvmlDeviceGetGpuInstanceProfileInfo (nvmlDevice_t device, unsigned int profile, nvmlGpuInstanceProfileInfo_t *info)

Parameters

device
The identifier of the target device
profile
One of the NVML_GPU_INSTANCE_PROFILE_*
info
Returns detailed profile information

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device, profile or info are invalid
- NVML_ERROR_NOT_SUPPORTED If device doesn't support MIG or profile isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation

Description

Get GPU instance profile information

Information provided by this API is immutable throughout the lifetime of a MIG mode.

For Ampere or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetGpuInstanceProfileInfoV (nvmlDevice_t device, unsigned int profile, nvmlGpuInstanceProfileInfo_v2_t *info)

Parameters

device
The identifier of the target device

profile
One of the NVML_GPU_INSTANCE_PROFILE_*

info
Returns detailed profile information

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device, profile, info, or info->version are invalid
- NVML_ERROR_NOT_SUPPORTED If device doesn't have MIG mode enabled or profile isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation

Description

Versioned wrapper around nvmlDeviceGetGpuInstanceProfileInfo that accepts a versioned nvmlGpuInstanceProfileInfo_v2_t or later output structure.

The caller must set the nvmlGpuInstanceProfileInfo_v2_t::version field to the appropriate version prior to calling this function. For example:

```
nvmlGpuInstanceProfileInfo_v2_t profileInfo =
            { .version = nvmlGpuInstanceProfileInfo_v2 };
      nvmlReturn_t result
= nvmlDeviceGetGpuInstanceProfileInfoV(device,

profile,

&profileInfo);
```

For Ampere or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetGpuInstancePossiblePlacements_v2

(nvmlDevice_t device, unsigned int profileId, nvmlGpuInstancePlacement_t *placements, unsigned int *count)

Parameters

device
The identifier of the target device

profileId
The GPU instance profile ID. See nvmlDeviceGetGpuInstanceProfileInfo

placements
Returns placements allowed for the profile. Can be NULL to discover number of allowed placements for this profile. If non-NULL must be large enough to accommodate the placements supported by the profile.

count
Returns number of allowed placemenets for the profile.

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device, profileId or count are invalid
- NVML_ERROR_NOT_SUPPORTED If device doesn't support MIG or profileId isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation

Description

Get GPU instance placements.

A placement represents the location of a GPU instance within a device. This API only returns all the possible placements for the given profile regardless of whether MIG is enabled or not. A created GPU instance occupies memory slices described by its placement. Creation of new GPU instance will fail if there is overlap with the already occupied memory slices.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

nvmlReturn_t nvmlDeviceGetGpuInstanceRemainingCapacity

(nvmlDevice_t device, unsigned int profileId, unsigned int *count)

Parameters

device
The identifier of the target device
profileId
The GPU instance profile ID. See nvmlDeviceGetGpuInstanceProfileInfo
count
Returns remaining instance count for the profile ID

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device, profileId or count are invalid
- NVML_ERROR_NOT_SUPPORTED If device doesn't have MIG mode enabled or profileId isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation

Description

Get GPU instance profile capacity.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

nvmlReturn_t nvmlDeviceCreateGpuInstance (nvmlDevice_t device, unsigned int profileId, nvmlGpuInstance_t *gpuInstance)

Parameters

device
The identifier of the target device
profileId
The GPU instance profile ID. See nvmlDeviceGetGpuInstanceProfileInfo
gpuInstance
Returns the GPU instance handle

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device, profile, profileId or gpuInstance are invalid
- NVML_ERROR_NOT_SUPPORTED If device doesn't have MIG mode enabled or in vGPU guest
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation
- NVML_ERROR_INSUFFICIENT_RESOURCES If the requested GPU instance could not be created

Description

Create GPU instance.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

If the parent device is unbound, reset or the GPU instance is destroyed explicitly, the GPU instance handle would become invalid. The GPU instance must be recreated to acquire a valid handle.

nvmlReturn_t nvmlDeviceCreateGpuInstanceWithPlacement (nvmlDevice_t device, unsigned int profileId, const nvmlGpuInstancePlacement_t *placement, nvmlGpuInstance_t *gpuInstance)

Parameters

device
The identifier of the target device

profileId
The GPU instance profile ID. See nvmlDeviceGetGpuInstanceProfileInfo

placement
The requested placement. See nvmlDeviceGetGpuInstancePossiblePlacements_v2

gpuInstance
Returns the GPU instance handle

Returns

- NVML_SUCCESS Upon success

- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device, profile, profileId, placement or gpuInstance are invalid
- NVML_ERROR_NOT_SUPPORTED If device doesn't have MIG mode enabled or in vGPU guest
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation
- NVML_ERROR_INSUFFICIENT_RESOURCES If the requested GPU instance could not be created

Description

Create GPU instance with the specified placement.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

If the parent device is unbound, reset or the GPU instance is destroyed explicitly, the GPU instance handle would become invalid. The GPU instance must be recreated to acquire a valid handle.

nvmlReturn_t nvmlGpuInstanceDestroy (nvmlGpuInstance_t gpuInstance)

Parameters

gpuInstance
The GPU instance handle

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If gpuInstance is invalid
- NVML_ERROR_NOT_SUPPORTED If device doesn't have MIG mode enabled or in vGPU guest
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation
- NVML_ERROR_IN_USE If the GPU instance is in use. This error would be returned if processes (e.g. CUDA application) or compute instances are active on the GPU instance.

Description

Destroy GPU instance.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

nvmlReturn_t nvmlDeviceGetGpuInstances (nvmlDevice_t device, unsigned int profileId, nvmlGpuInstance_t *gpuInstances, unsigned int *count)

Parameters

device
The identifier of the target device

profileId
The GPU instance profile ID. See nvmlDeviceGetGpuInstanceProfileInfo

gpuInstances
Returns pre-exiting GPU instances, the buffer must be large enough to accommodate the instances supported by the profile. See nvmlDeviceGetGpuInstanceProfileInfo

count
The count of returned GPU instances

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device, profileId, gpuInstances or count are invalid
- NVML_ERROR_NOT_SUPPORTED If device doesn't have MIG mode enabled
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation

Description

Get GPU instances for given profile ID.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

nvmlReturn_t nvmlDeviceGetGpuInstanceById (nvmlDevice_t device, unsigned int id, nvmlGpuInstance_t *gpuInstance)

Parameters

device
The identifier of the target device

id
The GPU instance ID

gpuInstance
Returns GPU instance

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device, id or gpuInstance are invalid
- NVML_ERROR_NOT_SUPPORTED If device doesn't have MIG mode enabled
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation
- NVML_ERROR_NOT_FOUND If the GPU instance is not found.

Description

Get GPU instances for given instance ID.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

nvmlReturn_t nvmlGpuInstanceGetInfo (nvmlGpuInstance_t gpuInstance, nvmlGpuInstanceInfo_t *info)

Parameters

gpuInstance
The GPU instance handle

info
Return GPU instance information

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If gpuInstance or info are invalid
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation

Description

Get GPU instance information.

For Ampere or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlGpuInstanceGetComputeInstanceProfileInfo (nvmlGpuInstance_t gpuInstance, unsigned int profile, unsigned int engProfile, nvmlComputeInstanceProfileInfo_t *info)

Parameters

gpuInstance
The identifier of the target GPU instance
profile
One of the NVML_COMPUTE_INSTANCE_PROFILE_*
engProfile
One of the NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_*
info
Returns detailed profile information

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If gpuInstance, profile, engProfile or info are invalid
- NVML_ERROR_NOT_SUPPORTED If profile isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation

Description

Get compute instance profile information.

Information provided by this API is immutable throughout the lifetime of a MIG mode.

For Ampere or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlGpuInstanceGetComputeInstanceProfileInfoV (nvmlGpuInstance_t gpuInstance, unsigned int profile, unsigned int engProfile, nvmlComputeInstanceProfileInfo_v2_t *info)

Parameters

gpuInstance
The identifier of the target GPU instance

profile
One of the NVML_COMPUTE_INSTANCE_PROFILE_*

engProfile
One of the NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_*

info
Returns detailed profile information

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If gpuInstance, profile, engProfile, info, or info->version are invalid
- NVML_ERROR_NOT_SUPPORTED If profile isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation

Description

Versioned wrapper around nvmlGpuInstanceGetComputeInstanceProfileInfo that accepts a versioned nvmlComputeInstanceProfileInfo_v2_t or later output structure.

The caller must set the nvmlGpuInstanceProfileInfo_v2_t::version field to the appropriate version prior to calling this function. For example:

```
    nvmlComputeInstanceProfileInfo_v2_t profileInfo =
            { .version = nvmlComputeInstanceProfileInfo_v2 };
        nvmlReturn_t result
= nvmlGpuInstanceGetComputeInstanceProfileInfoV(gpuInstance,

     profile,
```

```
        engProfile,
        &profileInfo);
```

For Ampere or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlGpuInstanceGetComputeInstanceRemainingCapacity (nvmlGpuInstance_t gpuInstance, unsigned int profileId, unsigned int *count)

Parameters

gpuInstance
The identifier of the target GPU instance

profileId
The compute instance profile ID. See nvmlGpuInstanceGetComputeInstanceProfileInfo

count
Returns remaining instance count for the profile ID

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If gpuInstance, profileId or availableCount are invalid
- NVML_ERROR_NOT_SUPPORTED If profileId isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation

Description

Get compute instance profile capacity.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

nvmlReturn_t nvmlGpuInstanceGetComputeInstancePossiblePlacements (nvmlGpuInstance_t gpuInstance, unsigned int profileId,

nvmlComputeInstancePlacement_t *placements, unsigned int *count)

Parameters

gpuInstance
The identifier of the target GPU instance

profileId
The compute instance profile ID. See nvmlGpuInstanceGetComputeInstanceProfileInfo

placements
Returns placements allowed for the profile. Can be NULL to discover number of allowed placements for this profile. If non-NULL must be large enough to accommodate the placements supported by the profile.

count
Returns number of allowed placemenets for the profile.

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If gpuInstance, profileId or count are invalid
- NVML_ERROR_NOT_SUPPORTED If device doesn't have MIG mode enabled or profileId isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation

Description

Get compute instance placements.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

A placement represents the location of a compute instance within a GPU instance. This API only returns all the possible placements for the given profile. A created compute instance occupies compute slices described by its placement. Creation of new compute instance will fail if there is overlap with the already occupied compute slices.

nvmlReturn_t nvmlGpuInstanceCreateComputeInstance (nvmlGpuInstance_t gpuInstance, unsigned int profileId, nvmlComputeInstance_t *computeInstance)

Parameters

gpuInstance
The identifier of the target GPU instance

profileId
The compute instance profile ID. See nvmlGpuInstanceGetComputeInstanceProfileInfo

computeInstance
Returns the compute instance handle

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If gpuInstance, profile, profileId or computeInstance are invalid
- NVML_ERROR_NOT_SUPPORTED If profileId isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation
- NVML_ERROR_INSUFFICIENT_RESOURCES If the requested compute instance could not be created

Description

Create compute instance.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

If the parent device is unbound, reset or the parent GPU instance is destroyed or the compute instance is destroyed explicitly, the compute instance handle would become invalid. The compute instance must be recreated to acquire a valid handle.

nvmlReturn_t nvmlGpuInstanceCreateComputeInstanceWithPlacement (nvmlGpuInstance_t gpuInstance, unsigned int profileId,

const nvmlComputeInstancePlacement_t *placement, nvmlComputeInstance_t *computeInstance)

Parameters

gpuInstance
The identifier of the target GPU instance

profileId
The compute instance profile ID. See nvmlGpuInstanceGetComputeInstanceProfileInfo

placement
The requested placement. See nvmlGpuInstanceGetComputeInstancePossiblePlacements

computeInstance
Returns the compute instance handle

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If gpuInstance, profile, profileId or computeInstance are invalid
- NVML_ERROR_NOT_SUPPORTED If profileId isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation
- NVML_ERROR_INSUFFICIENT_RESOURCES If the requested compute instance could not be created

Description

Create compute instance with the specified placement.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

If the parent device is unbound, reset or the parent GPU instance is destroyed or the compute instance is destroyed explicitly, the compute instance handle would become invalid. The compute instance must be recreated to acquire a valid handle.

nvmlReturn_t nvmlComputeInstanceDestroy (nvmlComputeInstance_t computeInstance)

Parameters

computeInstance
The compute instance handle

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If computeInstance is invalid
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation
- NVML_ERROR_IN_USE If the compute instance is in use. This error would be returned if processes (e.g. CUDA application) are active on the compute instance.

Description

Destroy compute instance.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

nvmlReturn_t nvmlGpuInstanceGetComputeInstances (nvmlGpuInstance_t gpuInstance, unsigned int profileId, nvmlComputeInstance_t *computeInstances, unsigned int *count)

Parameters

gpuInstance
The identifier of the target GPU instance
profileId
The compute instance profile ID. See nvmlGpuInstanceGetComputeInstanceProfileInfo
computeInstances
Returns pre-exiting compute instances, the buffer must be large enough to accommodate the instances supported by the profile. See nvmlGpuInstanceGetComputeInstanceProfileInfo

count
The count of returned compute instances

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If gpuInstance, profileId, computeInstances or count are invalid
- NVML_ERROR_NOT_SUPPORTED If profileId isn't supported
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation

Description

Get compute instances for given profile ID.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

nvmlReturn_t nvmlGpuInstanceGetComputeInstanceById (nvmlGpuInstance_t gpuInstance, unsigned int id, nvmlComputeInstance_t *computeInstance)

Parameters

gpuInstance
The identifier of the target GPU instance
id
The compute instance ID
computeInstance
Returns compute instance

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If device, ID or computeInstance are invalid
- NVML_ERROR_NOT_SUPPORTED If device doesn't have MIG mode enabled
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation
- NVML_ERROR_NOT_FOUND If the compute instance is not found.

Description

Get compute instance for given instance ID.

For Ampere or newer fully supported devices. Supported on Linux only. Requires privileged user.

nvmlReturn_t nvmlComputeInstanceGetInfo_v2 (nvmlComputeInstance_t computeInstance, nvmlComputeInstanceInfo_t *info)

Parameters

computeInstance
The compute instance handle

info
Return compute instance information

Returns

- NVML_SUCCESS Upon success
- NVML_ERROR_UNINITIALIZED If library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT If computeInstance or info are invalid
- NVML_ERROR_NO_PERMISSION If user doesn't have permission to perform the operation

Description

Get compute instance information.

For Ampere or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceIsMigDeviceHandle (nvmlDevice_t device, unsigned int *isMigDevice)

Parameters

device
NVML handle to test

isMigDevice
True when handle refers to a MIG device

Returns

- NVML_SUCCESS if device status was successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device handle or isMigDevice reference is invalid
- NVML_ERROR_NOT_SUPPORTED if this check is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Test if the given handle refers to a MIG device.

A MIG device handle is an NVML abstraction which maps to a MIG compute instance. These overloaded references can be used (with some restrictions) interchangeably with a GPU device handle to execute queries at a per-compute instance granularity.

For Ampere or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetGpuInstanceId (nvmlDevice_t device, unsigned int *id)

Parameters

device
Target MIG device handle

id
GPU instance ID

Returns

- NVML_SUCCESS if instance ID was successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device or id reference is invalid
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get GPU instance ID for the given MIG device handle.

GPU instance IDs are unique per device and remain valid until the GPU instance is destroyed.

For Ampere or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetComputeInstanceId (nvmlDevice_t device, unsigned int *id)

Parameters

device
Target MIG device handle

id
Compute instance ID

Returns

- NVML_SUCCESS if instance ID was successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device or id reference is invalid
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get compute instance ID for the given MIG device handle.

Compute instance IDs are unique per GPU instance and remain valid until the compute instance is destroyed.

For Ampere or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetMaxMigDeviceCount (nvmlDevice_t device, unsigned int *count)

Parameters

device
Target device handle

count
Count of MIG devices

Returns

- NVML_SUCCESS if count was successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device or count reference is invalid
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get the maximum number of MIG devices that can exist under a given parent NVML device.

Returns zero if MIG is not supported or enabled.

For Ampere or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetMigDeviceHandleByIndex (nvmlDevice_t device, unsigned int index, nvmlDevice_t *migDevice)

Parameters

device
Reference to the parent GPU device handle

index
Index of the MIG device

migDevice
Reference to the MIG device handle

Returns

- NVML_SUCCESS if migDevice handle was successfully created
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, index or migDevice reference is invalid
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_NOT_FOUND if no valid MIG device was found at index
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get MIG device handle for the given index under its parent NVML device.

If the compute instance is destroyed either explicitly or by destroying, resetting or unbinding the parent GPU instance or the GPU device itself the MIG device handle would remain invalid and must be requested again using this API. Handles may be reused and their properties can change in the process.

For Ampere or newer fully supported devices. Supported on Linux only.

nvmlReturn_t nvmlDeviceGetDeviceHandleFromMigDeviceHandle (nvmlDevice_t migDevice, nvmlDevice_t *device)

Parameters

migDevice
MIG device handle
device
Device handle

Returns

- NVML_SUCCESS if device handle was successfully created
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if migDevice or device is invalid
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get parent device handle from a MIG device handle.

For Ampere or newer fully supported devices. Supported on Linux only.

#define NVML_DEVICE_MIG_DISABLE 0x0

Disable Multi Instance GPU mode.

#define NVML_DEVICE_MIG_ENABLE 0x1

Enable Multi Instance GPU mode.

#define NVML_GPU_INSTANCE_PROFILE_1_SLICE 0x0

GPU instance profiles.

These macros should be passed to nvmlDeviceGetGpuInstanceProfileInfo to retrieve the detailed information about a GPU instance such as profile ID, engine counts.

#define NVML_GPU_INTSTANCE_PROFILE_CAPS_P2P 0x1

MIG GPU instance profile capability.

Bit field values representing MIG profile capabilities
nvmlGpuInstanceProfileInfo_v3_t::capabilities

#define nvmlGpuInstanceProfileInfo_v2 NVML_STRUCT_VERSION(GpuInstanceProfileInfo, 2)

Version identifier value for nvmlGpuInstanceProfileInfo_v2_t::version.

#define nvmlGpuInstanceProfileInfo_v3 NVML_STRUCT_VERSION(GpuInstanceProfileInfo, 3)

Version identifier value for nvmlGpuInstanceProfileInfo_v3_t::version.

#define NVML_COMPUTE_INSTANCE_PROFILE_1_SLICE 0x0

Compute instance profiles.

These macros should be passed to nvmlGpuInstanceGetComputeInstanceProfileInfo to retrieve the detailed information about a compute instance such as profile ID, engine counts

#define NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_SHARED 0x0

All the engines except multiprocessors would be shared.

#define nvmlComputeInstanceProfileInfo_v2 NVML_STRUCT_VERSION(ComputeInstanceProfileInfo, 2)

Version identifier value for nvmlComputeInstanceProfileInfo_v2_t::version.

#define nvmlComputeInstanceProfileInfo_v3 NVML_STRUCT_VERSION(ComputeInstanceProfileInfo, 3)

Version identifier value for nvmlComputeInstanceProfileInfo_v3_t::version.

4.29. NVML GPM

GPM Enums

GPM Structs

GPM Functions

4.29.1. GPM Enums

NVML GPM

enum nvmlGpmMetricId_t

GPM Metric Identifiers

Values

NVML_GPM_METRIC_GRAPHICS_UTIL = 1
Percentage of time any compute/graphics app was active on the GPU. 0.0 - 100.0.
NVML_GPM_METRIC_SM_UTIL = 2
Percentage of SMs that were busy. 0.0 - 100.0.
NVML_GPM_METRIC_SM_OCCUPANCY = 3
Percentage of warps that were active vs theoretical maximum. 0.0 - 100.0.
NVML_GPM_METRIC_INTEGER_UTIL = 4
Percentage of time the GPU's SMs were doing integer operations. 0.0 - 100.0.
NVML_GPM_METRIC_ANY_TENSOR_UTIL = 5
Percentage of time the GPU's SMs were doing ANY tensor operations. 0.0 - 100.0.
NVML_GPM_METRIC_DFMA_TENSOR_UTIL = 6
Percentage of time the GPU's SMs were doing DFMA tensor operations. 0.0 - 100.0.
NVML_GPM_METRIC_HMMA_TENSOR_UTIL = 7
Percentage of time the GPU's SMs were doing HMMA tensor operations. 0.0 - 100.0.
NVML_GPM_METRIC_IMMA_TENSOR_UTIL = 9
Percentage of time the GPU's SMs were doing IMMA tensor operations. 0.0 - 100.0.
NVML_GPM_METRIC_DRAM_BW_UTIL = 10
Percentage of DRAM bw used vs theoretical maximum. 0.0 - 100.0 */.
NVML_GPM_METRIC_FP64_UTIL = 11
Percentage of time the GPU's SMs were doing non-tensor FP64 math. 0.0 - 100.0.
NVML_GPM_METRIC_FP32_UTIL = 12
Percentage of time the GPU's SMs were doing non-tensor FP32 math. 0.0 - 100.0.
NVML_GPM_METRIC_FP16_UTIL = 13
Percentage of time the GPU's SMs were doing non-tensor FP16 math. 0.0 - 100.0.
NVML_GPM_METRIC_PCIE_TX_PER_SEC = 20
PCIe traffic from this GPU in MiB/sec.

NVML_GPM_METRIC_PCIE_RX_PER_SEC = 21
PCIe traffic to this GPU in MiB/sec.
NVML_GPM_METRIC_NVDEC_0_UTIL = 30
Percent utilization of NVDEC 0. 0.0 - 100.0.
NVML_GPM_METRIC_NVDEC_1_UTIL = 31
Percent utilization of NVDEC 1. 0.0 - 100.0.
NVML_GPM_METRIC_NVDEC_2_UTIL = 32
Percent utilization of NVDEC 2. 0.0 - 100.0.
NVML_GPM_METRIC_NVDEC_3_UTIL = 33
Percent utilization of NVDEC 3. 0.0 - 100.0.
NVML_GPM_METRIC_NVDEC_4_UTIL = 34
Percent utilization of NVDEC 4. 0.0 - 100.0.
NVML_GPM_METRIC_NVDEC_5_UTIL = 35
Percent utilization of NVDEC 5. 0.0 - 100.0.
NVML_GPM_METRIC_NVDEC_6_UTIL = 36
Percent utilization of NVDEC 6. 0.0 - 100.0.
NVML_GPM_METRIC_NVDEC_7_UTIL = 37
Percent utilization of NVDEC 7. 0.0 - 100.0.
NVML_GPM_METRIC_NVJPG_0_UTIL = 40
Percent utilization of NVJPG 0. 0.0 - 100.0.
NVML_GPM_METRIC_NVJPG_1_UTIL = 41
Percent utilization of NVJPG 1. 0.0 - 100.0.
NVML_GPM_METRIC_NVJPG_2_UTIL = 42
Percent utilization of NVJPG 2. 0.0 - 100.0.
NVML_GPM_METRIC_NVJPG_3_UTIL = 43
Percent utilization of NVJPG 3. 0.0 - 100.0.
NVML_GPM_METRIC_NVJPG_4_UTIL = 44
Percent utilization of NVJPG 4. 0.0 - 100.0.
NVML_GPM_METRIC_NVJPG_5_UTIL = 45
Percent utilization of NVJPG 5. 0.0 - 100.0.
NVML_GPM_METRIC_NVJPG_6_UTIL = 46
Percent utilization of NVJPG 6. 0.0 - 100.0.
NVML_GPM_METRIC_NVJPG_7_UTIL = 47
Percent utilization of NVJPG 7. 0.0 - 100.0.
NVML_GPM_METRIC_NVOFA_0_UTIL = 50
Percent utilization of NVOFA 0. 0.0 - 100.0.
NVML_GPM_METRIC_NVLINK_TOTAL_RX_PER_SEC = 60
NvLink read bandwidth for all links in MiB/sec.
NVML_GPM_METRIC_NVLINK_TOTAL_TX_PER_SEC = 61
NvLink write bandwidth for all links in MiB/sec.
NVML_GPM_METRIC_NVLINK_L0_RX_PER_SEC = 62
NvLink read bandwidth for link 0 in MiB/sec.

NVML_GPM_METRIC_NVLINK_L0_TX_PER_SEC = 63
NvLink write bandwidth for link 0 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L1_RX_PER_SEC = 64
NvLink read bandwidth for link 1 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L1_TX_PER_SEC = 65
NvLink write bandwidth for link 1 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L2_RX_PER_SEC = 66
NvLink read bandwidth for link 2 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L2_TX_PER_SEC = 67
NvLink write bandwidth for link 2 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L3_RX_PER_SEC = 68
NvLink read bandwidth for link 3 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L3_TX_PER_SEC = 69
NvLink write bandwidth for link 3 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L4_RX_PER_SEC = 70
NvLink read bandwidth for link 4 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L4_TX_PER_SEC = 71
NvLink write bandwidth for link 4 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L5_RX_PER_SEC = 72
NvLink read bandwidth for link 5 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L5_TX_PER_SEC = 73
NvLink write bandwidth for link 5 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L6_RX_PER_SEC = 74
NvLink read bandwidth for link 6 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L6_TX_PER_SEC = 75
NvLink write bandwidth for link 6 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L7_RX_PER_SEC = 76
NvLink read bandwidth for link 7 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L7_TX_PER_SEC = 77
NvLink write bandwidth for link 7 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L8_RX_PER_SEC = 78
NvLink read bandwidth for link 8 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L8_TX_PER_SEC = 79
NvLink write bandwidth for link 8 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L9_RX_PER_SEC = 80
NvLink read bandwidth for link 9 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L9_TX_PER_SEC = 81
NvLink write bandwidth for link 9 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L10_RX_PER_SEC = 82
NvLink read bandwidth for link 10 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L10_TX_PER_SEC = 83
NvLink write bandwidth for link 10 in MiB/sec.

NVML_GPM_METRIC_NVLINK_L11_RX_PER_SEC = 84
NvLink read bandwidth for link 11 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L11_TX_PER_SEC = 85
NvLink write bandwidth for link 11 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L12_RX_PER_SEC = 86
NvLink read bandwidth for link 12 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L12_TX_PER_SEC = 87
NvLink write bandwidth for link 12 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L13_RX_PER_SEC = 88
NvLink read bandwidth for link 13 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L13_TX_PER_SEC = 89
NvLink write bandwidth for link 13 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L14_RX_PER_SEC = 90
NvLink read bandwidth for link 14 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L14_TX_PER_SEC = 91
NvLink write bandwidth for link 14 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L15_RX_PER_SEC = 92
NvLink read bandwidth for link 15 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L15_TX_PER_SEC = 93
NvLink write bandwidth for link 15 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L16_RX_PER_SEC = 94
NvLink read bandwidth for link 16 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L16_TX_PER_SEC = 95
NvLink write bandwidth for link 16 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L17_RX_PER_SEC = 96
NvLink read bandwidth for link 17 in MiB/sec.
NVML_GPM_METRIC_NVLINK_L17_TX_PER_SEC = 97
NvLink write bandwidth for link 17 in MiB/sec.
NVML_GPM_METRIC_MAX = 98
Maximum value above +1. Note that changing this should also change NVML_GPM_METRICS_GET_VERSION due to struct size change.

4.29.2. GPM Structs

NVML GPM

struct nvmlGpmMetric_t

struct nvmlGpmMetricsGet_t

struct nvmlGpmSupport_t

typedef struct nvmlGpmSample_st *nvmlGpmSample_t

Handle to an allocated GPM sample allocated with nvmlGpmSampleAlloc(). Free this with nvmlGpmSampleFree().

4.29.3. GPM Functions

NVML GPM

nvmlReturn_t nvmlGpmMetricsGet (nvmlGpmMetricsGet_t *metricsGet)

Parameters

metricsGet
IN/OUT: populated nvmlGpmMetricsGet_t struct

Returns

- NVML_SUCCESS on success
- Nonzero NVML_ERROR_? enum on error

Description

Calculate GPM metrics from two samples.

For Hopper or newer fully supported devices.

nvmlReturn_t nvmlGpmSampleFree (nvmlGpmSample_t gpmSample)

Parameters

gpmSample
Sample to free

Returns

- NVML_SUCCESS on success
- NVML_ERROR_INVALID_ARGUMENT if an invalid pointer is provided

Description

Free an allocated sample buffer that was allocated with nvmlGpmSampleAlloc()

For Hopper or newer fully supported devices.

nvmlReturn_t nvmlGpmSampleAlloc (nvmlGpmSample_t *gpmSample)

Parameters

gpmSample
Where the allocated sample will be stored

Returns

- NVML_SUCCESS on success
- NVML_ERROR_INVALID_ARGUMENT if an invalid pointer is provided
- NVML_ERROR_MEMORY if system memory is insufficient

Description

Allocate a sample buffer to be used with NVML GPM . You will need to allocate at least two of these buffers to use with the NVML GPM feature

For Hopper or newer fully supported devices.

nvmlReturn_t nvmlGpmSampleGet (nvmlDevice_t device, nvmlGpmSample_t gpmSample)

Parameters

device
Device to get samples for
gpmSample
Buffer to read samples into

Returns

- NVML_SUCCESS on success
- Nonzero NVML_ERROR_? enum on error

Description

Read a sample of GPM metrics into the provided gpmSample buffer. After two samples are gathered, you can call nvmlGpmMetricGet on those samples to retrive metrics

For Hopper or newer fully supported devices.

nvmlReturn_t nvmlGpmMigSampleGet (nvmlDevice_t device, unsigned int gpuInstanceId, nvmlGpmSample_t gpmSample)

Parameters

device
Device to get samples for
gpuInstanceId
MIG GPU Instance ID
gpmSample
Buffer to read samples into

Returns

- NVML_SUCCESS on success
- Nonzero NVML_ERROR_? enum on error

Description

Read a sample of GPM metrics into the provided gpmSample buffer for a MIG GPU Instance.

After two samples are gathered, you can call nvmlGpmMetricGet on those samples to retrive metrics

For Hopper or newer fully supported devices.

nvmlReturn_t nvmlGpmQueryDeviceSupport (nvmlDevice_t device, nvmlGpmSupport_t *gpmSupport)

Parameters

device
NVML device to query for
gpmSupport
Structure to indicate GPM support nvmlGpmSupport_t. Indicates GPM support per system for the supplied device

Returns

- NVML_SUCCESS on success
- Nonzero NVML_ERROR_? enum if there is an error in processing the query

Description

Indicate whether the supplied device supports GPM

nvmlReturn_t nvmlGpmQueryIfStreamingEnabled (nvmlDevice_t device, unsigned int *state)

Parameters

device
The identifier of the target device

state
Returns GPM stream state NVML_FEATURE_DISABLED or NVML_FEATURE_ENABLED

Returns

- NVML_SUCCESS if current GPM stream state were successfully queried
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or state is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device

Description

Get GPM stream state.

HOPPER_OR_NEWER% Supported on Linux, Windows TCC.

nvmlReturn_t nvmlGpmSetStreamingEnabled (nvmlDevice_t device, unsigned int state)

Parameters

device
The identifier of the target device

state
GPM stream state, NVML_FEATURE_DISABLED or NVML_FEATURE_ENABLED

Returns

- NVML_SUCCESS if current GPM stream state is successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device

Description

Set GPM stream state.

HOPPER_OR_NEWER% Supported on Linux, Windows TCC.

4.30. VirtualGPU

vGPU Enums

vGPU Constants

vGPU Structs

4.30.1. vGPU Enums

VirtualGPU

enum nvmlGpuVirtualizationMode_t

GPU virtualization mode types.

Values

NVML_GPU_VIRTUALIZATION_MODE_NONE = 0
Represents Bare Metal GPU.

NVML_GPU_VIRTUALIZATION_MODE_PASSTHROUGH = 1
Device is associated with GPU-Passthorugh.

NVML_GPU_VIRTUALIZATION_MODE_VGPU = 2
Device is associated with vGPU inside virtual machine.

NVML_GPU_VIRTUALIZATION_MODE_HOST_VGPU = 3
Device is associated with VGX hypervisor in vGPU mode.

NVML_GPU_VIRTUALIZATION_MODE_HOST_VSGA = 4
Device is associated with VGX hypervisor in vSGA mode.

enum nvmlHostVgpuMode_t

Host vGPU modes

Values

NVML_HOST_VGPU_MODE_NON_SRIOV = 0
Non SR-IOV mode.

NVML_HOST_VGPU_MODE_SRIOV = 1
SR-IOV mode.

enum nvmlVgpuVmIdType_t

Types of VM identifiers

Values

NVML_VGPU_VM_ID_DOMAIN_ID = 0
VM ID represents DOMAIN ID.
NVML_VGPU_VM_ID_UUID = 1
VM ID represents UUID.

enum nvmlVgpuGuestInfoState_t

vGPU GUEST info state

Values

NVML_VGPU_INSTANCE_GUEST_INFO_STATE_UNINITIALIZED = 0
Guest-dependent fields uninitialized.
NVML_VGPU_INSTANCE_GUEST_INFO_STATE_INITIALIZED = 1
Guest-dependent fields initialized.

enum nvmlGridLicenseFeatureCode_t

vGPU software licensable features

Values

NVML_GRID_LICENSE_FEATURE_CODE_UNKNOWN = 0
Unknown.
NVML_GRID_LICENSE_FEATURE_CODE_VGPU = 1
Virtual GPU.
NVML_GRID_LICENSE_FEATURE_CODE_NVIDIA_RTX = 2
Nvidia RTX.
NVML_GRID_LICENSE_FEATURE_CODE_VWORKSTATION =
NVML_GRID_LICENSE_FEATURE_CODE_NVIDIA_RTX
Deprecated, do not use.
NVML_GRID_LICENSE_FEATURE_CODE_GAMING = 3
Gaming.
NVML_GRID_LICENSE_FEATURE_CODE_COMPUTE = 4
Compute.

enum nvmlVgpuCapability_t

vGPU queryable capabilities

Values

NVML_VGPU_CAP_NVLINK_P2P = 0
P2P over NVLink is supported.
NVML_VGPU_CAP_GPUDIRECT = 1
GPUDirect capability is supported.
NVML_VGPU_CAP_MULTI_VGPU_EXCLUSIVE = 2
vGPU profile cannot be mixed with other vGPU profiles in same VM
NVML_VGPU_CAP_EXCLUSIVE_TYPE = 3
vGPU profile cannot run on a GPU alongside other profiles of different type
NVML_VGPU_CAP_EXCLUSIVE_SIZE = 4
vGPU profile cannot run on a GPU alongside other profiles of different size
NVML_VGPU_CAP_COUNT

enum nvmlVgpuDriverCapability_t

vGPU driver queryable capabilities

Values

NVML_VGPU_DRIVER_CAP_HETEROGENEOUS_MULTI_VGPU = 0
Supports mixing of different vGPU profiles within one guest VM.
NVML_VGPU_DRIVER_CAP_COUNT

enum nvmlDeviceVgpuCapability_t

Device vGPU queryable capabilities

Values

NVML_DEVICE_VGPU_CAP_FRACTIONAL_MULTI_VGPU = 0
Query if the fractional vGPU profiles on this GPU can be used in multi-vGPU configurations.
NVML_DEVICE_VGPU_CAP_HETEROGENEOUS_TIMESLICE_PROFILES = 1
Query if the GPU support concurrent execution of timesliced vGPU profiles of differing types.
NVML_DEVICE_VGPU_CAP_HETEROGENEOUS_TIMESLICE_SIZES = 2
Query if the GPU support concurrent execution of timesliced vGPU profiles of differing framebuffer sizes.
NVML_DEVICE_VGPU_CAP_READ_DEVICE_BUFFER_BW = 3
Query the GPU's read_device_buffer expected bandwidth capacity in megabytes per second.

NVML_DEVICE_VGPU_CAP_WRITE_DEVICE_BUFFER_BW = 4
Query the GPU's write_device_buffer expected bandwidth capacity in megabytes per second.
NVML_DEVICE_VGPU_CAP_DEVICE_STREAMING = 5
Query if vGPU profiles on the GPU supports migration data streaming.
NVML_DEVICE_VGPU_CAP_MINI_QUARTER_GPU = 6
Set/Get support for mini-quarter vGPU profiles.
NVML_DEVICE_VGPU_CAP_COMPUTE_MEDIA_ENGINE_GPU = 7
Set/Get support for compute media engine vGPU profiles.
NVML_DEVICE_VGPU_CAP_COUNT

#define NVML_GRID_LICENSE_EXPIRY_NOT_AVAILABLE 0

Expiry information not available.

Status codes for license expiry

#define NVML_GRID_LICENSE_EXPIRY_INVALID 1

Invalid expiry or error fetching expiry.

#define NVML_GRID_LICENSE_EXPIRY_VALID 2

Valid expiry.

#define NVML_GRID_LICENSE_EXPIRY_NOT_APPLICABLE 3

Expiry not applicable.

#define NVML_GRID_LICENSE_EXPIRY_PERMANENT 4

Permanent expiry.

4.30.2. vGPU Constants

VirtualGPU

#define NVML_GRID_LICENSE_BUFFER_SIZE 128

Buffer size guaranteed to be large enough for nvmlVgpuTypeGetLicense

#define NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION 0:0

Macros for vGPU instance's virtualization capabilities bitfield.

#define NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION 0:0

Macros for pGPU's virtualization capabilities bitfield.

4.30.3. vGPU Structs

VirtualGPU

struct nvmlVgpuHeterogeneousMode_v1_t

struct nvmlVgpuPlacementId_v1_t

struct nvmlVgpuPlacementList_v1_t

struct nvmlVgpuInstanceUtilizationSample_t

struct nvmlVgpuInstanceUtilizationInfo_v1_t

struct nvmlVgpuInstancesUtilizationInfo_v1_t

struct nvmlVgpuProcessUtilizationSample_t

struct nvmlVgpuProcessUtilizationInfo_v1_t

struct nvmlVgpuProcessesUtilizationInfo_v1_t

union nvmlVgpuSchedulerParams_t

struct nvmlVgpuSchedulerLogEntry_t

struct nvmlVgpuSchedulerLog_t

struct nvmlVgpuSchedulerGetState_t

union nvmlVgpuSchedulerSetParams_t

struct nvmlVgpuSchedulerSetState_t

struct nvmlVgpuSchedulerCapabilities_t

struct nvmlVgpuLicenseExpiry_t

struct nvmlProcessUtilizationSample_t

struct nvmlProcessUtilizationInfo_v1_t

struct nvmlProcessesUtilizationInfo_v1_t

struct nvmlGridLicenseExpiry_t

struct nvmlGridLicensableFeature_t

struct nvmlGridLicensableFeatures_t

struct nvmlEccSramErrorStatus_v1_t

#define NVML_VGPU_SCHEDULER_POLICY_UNKNOWN 0

vGPU scheduler policies

#define NVML_GRID_LICENSE_STATE_UNKNOWN 0

Unknown state.

vGPU license state

#define NVML_GRID_LICENSE_STATE_UNINITIALIZED 1

Uninitialized state.

#define NVML_GRID_LICENSE_STATE_UNLICENSED_UNRESTRICTED 2

Unlicensed unrestricted state.

#define NVML_GRID_LICENSE_STATE_UNLICENSED_RESTRICTED 3

Unlicensed restricted state.

#define NVML_GRID_LICENSE_STATE_UNLICENSED 4

Unlicensed state.

#define NVML_GRID_LICENSE_STATE_LICENSED 5

Licensed state.

#define NVML_GSP_FIRMWARE_VERSION_BUF_SIZE 0x40

GSP firmware

#define NVML_DEVICE_ARCH_KEPLER 2

Simplified chip architecture

#define NVML_BUS_TYPE_UNKNOWN 0

PCI bus types

#define NVML_FAN_POLICY_TEMPERATURE_CONTINOUS_SW 0

Device Power Modes Device Fan control policy

#define NVML_POWER_SOURCE_AC 0x00000000

Device Power Source

4.31. NvmlClocksEventReasons

#define nvmlClocksEventReasonGpuIdle 0x0000000000000001LL

Nothing is running on the GPU and the clocks are dropping to Idle state

This limiter may be removed in a later release

#define nvmlClocksEventReasonApplicationsClocksSetting 0x0000000000000002LL

GPU clocks are limited by current setting of applications clocks

See also:

nvmlDeviceSetApplicationsClocks

nvmlDeviceGetApplicationsClock

#define nvmlClocksThrottleReasonUserDefinedClocks nvmlClocksEventReasonApplicationsClocksSetting

Deprecated Renamed to nvmlClocksThrottleReasonApplicationsClocksSetting as the name describes the situation more accurately.

#define nvmlClocksEventReasonSwPowerCap 0x0000000000000004LL

The clocks have been optimized to ensure not to exceed currently set power limits

See also:

nvmlDeviceGetPowerUsage

nvmlDeviceSetPowerManagementLimit

nvmlDeviceGetPowerManagementLimit

#define nvmlClocksThrottleReasonHwSlowdown 0x0000000000000008LL

HW Slowdown (reducing the core clocks by a factor of 2 or more) is engaged

This is an indicator of:

- temperature being too high
- External Power Brake Assertion is triggered (e.g. by the system power supply)
- Power draw is too high and Fast Trigger protection is reducing the clocks
- May be also reported during PState or clock change
  - This behavior may be removed in a later release.

See also:

nvmlDeviceGetTemperature

nvmlDeviceGetTemperatureThreshold

nvmlDeviceGetPowerUsage

#define nvmlClocksEventReasonSyncBoost 0x0000000000000010LL

Sync Boost

This GPU has been added to a Sync boost group with nvidia-smi or DCGM in order to maximize performance per watt. All GPUs in the sync boost group will boost to the minimum possible clocks across the entire group. Look at the throttle reasons for other GPUs in the system to see why those GPUs are holding this one at lower clocks.

#define nvmlClocksEventReasonSwThermalSlowdown 0x0000000000000020LL

SW Thermal Slowdown

The current clocks have been optimized to ensure the the following is true:

- Current GPU temperature does not exceed GPU Max Operating Temperature

- Current memory temperature does not exceeed Memory Max Operating Temperature

#define nvmlClocksThrottleReasonHwThermalSlowdown 0x0000000000000040LL

HW Thermal Slowdown (reducing the core clocks by a factor of 2 or more) is engaged

This is an indicator of:

- temperature being too high

See also:

nvmlDeviceGetTemperature

nvmlDeviceGetTemperatureThreshold

nvmlDeviceGetPowerUsage

#define nvmlClocksThrottleReasonHwPowerBrakeSlowdown 0x0000000000000080LL

HW Power Brake Slowdown (reducing the core clocks by a factor of 2 or more) is engaged

This is an indicator of:

- External Power Brake Assertion being triggered (e.g. by the system power supply)

See also:

nvmlDeviceGetTemperature

nvmlDeviceGetTemperatureThreshold

nvmlDeviceGetPowerUsage

#define nvmlClocksEventReasonDisplayClockSetting 0x0000000000000100LL

GPU clocks are limited by current setting of Display clocks

See also:

bug 1997531

#define nvmlClocksEventReasonNone 0x0000000000000000LL

Bit mask representing no clocks throttling

Clocks are as high as possible.

#define nvmlClocksEventReasonAll (nvmlClocksThrottleReasonNone \ | nvmlClocksEventReasonGpuIdle \ | nvmlClocksEventReasonApplicationsClocksSetting \ | nvmlClocksEventReasonSwPowerCap \ | nvmlClocksThrottleReasonHwSlowdown \ | nvmlClocksEventReasonSyncBoost \ | nvmlClocksEventReasonSwThermalSlowdown \ | nvmlClocksThrottleReasonHwThermalSlowdown \ | nvmlClocksThrottleReasonHwPowerBrakeSlowdown \ | nvmlClocksEventReasonDisplayClockSetting \ )

Bit mask representing all supported clocks throttling reasons New reasons might be added to this list in the future

#define nvmlClocksThrottleReasonGpuIdle nvmlClocksEventReasonGpuIdle

Deprecated Use nvmlClocksEventReasonGpuIdle instead

#define nvmlClocksThrottleReasonApplicationsClocksSetting nvmlClocksEventReasonApplicationsClocksSetting

Deprecated Use nvmlClocksEventReasonApplicationsClocksSetting instead

#define nvmlClocksThrottleReasonSyncBoost nvmlClocksEventReasonSyncBoost

Deprecated Use nvmlClocksEventReasonSyncBoost instead

#define nvmlClocksThrottleReasonSwPowerCap nvmlClocksEventReasonSwPowerCap

Deprecated Use nvmlClocksEventReasonSwPowerCap instead

#define nvmlClocksThrottleReasonSwThermalSlowdown nvmlClocksEventReasonSwThermalSlowdown

Deprecated Use nvmlClocksEventReasonSwThermalSlowdown instead

#define nvmlClocksThrottleReasonDisplayClockSetting nvmlClocksEventReasonDisplayClockSetting

Deprecated Use nvmlClocksEventReasonDisplayClockSetting instead

#define nvmlClocksThrottleReasonNone nvmlClocksEventReasonNone

Deprecated Use nvmlClocksEventReasonNone instead

#define nvmlClocksThrottleReasonAll nvmlClocksEventReasonAll

Deprecated Use nvmlClocksEventReasonAll instead

Chapter 5. DATA STRUCTURES

Here are the data structures with brief descriptions:

nvmlAccountingStats_t
nvmlBAR1Memory_t
nvmlBridgeChipHierarchy_t
nvmlBridgeChipInfo_t
nvmlC2cModeInfo_v1_t
nvmlClkMonFaultInfo_t
nvmlClkMonStatus_t
nvmlComputeInstanceProfileInfo_t
nvmlComputeInstanceProfileInfo_v2_t
nvmlComputeInstanceProfileInfo_v3_t
nvmlConfComputeMemSizeInfo_t
nvmlEccErrorCounts_t
nvmlEccSramErrorStatus_v1_t
nvmlEncoderSessionInfo_t
nvmlEventData_t
nvmlExcludedDeviceInfo_t
nvmlFBCSessionInfo_t
nvmlFBCStats_t
nvmlFieldValue_t
nvmlGpmMetric_t
nvmlGpmMetricsGet_t
nvmlGpmSupport_t
nvmlGpuFabricInfo_v2_t
nvmlGpuInstancePlacement_t
nvmlGpuInstanceProfileInfo_t
nvmlGpuInstanceProfileInfo_v2_t
nvmlGpuInstanceProfileInfo_v3_t
nvmlGridLicensableFeature_t
nvmlGridLicensableFeatures_t

nvmlVgpuSchedulerSetState_t
nvmlVgpuVersion_t
nvmlViolationTime_t

5.1. nvmlAccountingStats_t Struct Reference

Describes accounting statistics of a process.

unsigned int nvmlAccountingStats_t::gpuUtilization

Description

Percent of time over the process's lifetime during which one or more kernels was executing on the GPU. Utilization stats just like returned by nvmlDeviceGetUtilizationRates but for the life time of a process (not just the last sample period). Set to NVML_VALUE_NOT_AVAILABLE if nvmlDeviceGetUtilizationRates is not supported

unsigned int nvmlAccountingStats_t::memoryUtilization

Description

Percent of time over the process's lifetime during which global (device) memory was being read or written. Set to NVML_VALUE_NOT_AVAILABLE if nvmlDeviceGetUtilizationRates is not supported

unsigned long long nvmlAccountingStats_t::maxMemoryUsage

Description

Maximum total memory in bytes that was ever allocated by the process. Set to NVML_VALUE_NOT_AVAILABLE if nvmlProcessInfo_t->usedGpuMemory is not supported

unsigned long long nvmlAccountingStats_t::time

Description

Amount of time in ms during which the compute context was active. The time is reported as 0 if the process is not terminated

unsigned long long nvmlAccountingStats_t::startTime

CPU Timestamp in usec representing start time for the process.

unsigned int nvmlAccountingStats_t::isRunning

Flag to represent if the process is running (1 for running, 0 for terminated).

unsigned int nvmlAccountingStats_t::reserved

Reserved for future use.

5.2. nvmlBAR1Memory_t Struct Reference

BAR1 Memory allocation Information for a device

unsigned long long nvmlBAR1Memory_t::bar1Total

Total BAR1 Memory (in bytes).

unsigned long long nvmlBAR1Memory_t::bar1Free

Unallocated BAR1 Memory (in bytes).

unsigned long long nvmlBAR1Memory_t::bar1Used

Allocated Used Memory (in bytes).

5.3. nvmlBridgeChipHierarchy_t Struct Reference

This structure stores the complete Hierarchy of the Bridge Chip within the board. The immediate bridge is stored at index 0 of bridgeInfoList, parent to immediate bridge is at index 1 and so forth.

unsigned char nvmlBridgeChipHierarchy_t::bridgeCount

Number of Bridge Chips on the Board.

struct nvmlBridgeChipInfo_t nvmlBridgeChipHierarchy_t::bridgeChipInfo

Hierarchy of Bridge Chips on the board.

5.4. nvmlBridgeChipInfo_t Struct Reference

Information about the Bridge Chip Firmware

nvmlBridgeChipType_t nvmlBridgeChipInfo_t::type

Type of Bridge Chip.

unsigned int nvmlBridgeChipInfo_t::fwVersion

Firmware Version. 0=Version is unavailable.

5.5. nvmlC2cModeInfo_v1_t Struct Reference

C2C Mode information for a device

5.6. nvmlClkMonFaultInfo_t Struct Reference

Clock Monitor error types

unsigned int nvmlClkMonFaultInfo_t::clkApiDomain

Description

The Domain which faulted

unsigned int nvmlClkMonFaultInfo_t::clkDomainFaultMask

Description

Faults Information

5.7. nvmlClkMonStatus_t Struct Reference

Clock Monitor Status

unsigned int nvmlClkMonStatus_t::bGlobalStatus

Description

Fault status Indicator

unsigned int nvmlClkMonStatus_t::clkMonListSize

Description

Total faulted domain numbers

struct nvmlClkMonFaultInfo_t nvmlClkMonStatus_t::clkMonList

Description

The fault Information structure

5.8. nvmlComputeInstanceProfileInfo_t Struct Reference

Compute instance profile information.

unsigned int nvmlComputeInstanceProfileInfo_t::id

Unique profile ID within the GPU instance.

unsigned int nvmlComputeInstanceProfileInfo_t::sliceCount

GPU Slice count.

unsigned int nvmlComputeInstanceProfileInfo_t::instanceCount

Compute instance count.

unsigned int nvmlComputeInstanceProfileInfo_t::multiprocessorCount

Streaming Multiprocessor count.

unsigned int nvmlComputeInstanceProfileInfo_t::sharedCopyEngineCount

Shared Copy Engine count.

unsigned int nvmlComputeInstanceProfileInfo_t::sharedDecoderCount

Shared Decoder Engine count.

unsigned int nvmlComputeInstanceProfileInfo_t::sharedEncoderCount

Shared Encoder Engine count.

unsigned int nvmlComputeInstanceProfileInfo_t::sharedJpegCount

Shared JPEG Engine count.

unsigned int nvmlComputeInstanceProfileInfo_t::sharedOfaCount

Shared OFA Engine count.

5.9. nvmlComputeInstanceProfileInfo_v2_t Struct Reference

Compute instance profile information (v2).

Version 2 adds the nvmlComputeInstanceProfileInfo_v2_t::version field to the start of the structure, and the nvmlComputeInstanceProfileInfo_v2_t::name field to the end. This structure is not backwards-compatible with nvmlComputeInstanceProfileInfo_t.

unsigned int nvmlComputeInstanceProfileInfo_v2_t::version

Structure version identifier (set to nvmlComputeInstanceProfileInfo_v2).

unsigned int nvmlComputeInstanceProfileInfo_v2_t::id

Unique profile ID within the GPU instance.

unsigned int nvmlComputeInstanceProfileInfo_v2_t::sliceCount

GPU Slice count.

unsigned int nvmlComputeInstanceProfileInfo_v2_t::instanceCount

Compute instance count.

unsigned int nvmlComputeInstanceProfileInfo_v2_t::multiprocessorCount

Streaming Multiprocessor count.

unsigned int nvmlComputeInstanceProfileInfo_v2_t::sharedCopyEngineCount

Shared Copy Engine count.

unsigned int nvmlComputeInstanceProfileInfo_v2_t::sharedDecoderCount

Shared Decoder Engine count.

unsigned int nvmlComputeInstanceProfileInfo_v2_t::sharedEncoderCount

Shared Encoder Engine count.

unsigned int nvmlComputeInstanceProfileInfo_v2_t::sharedJpegCount

Shared JPEG Engine count.

unsigned int nvmlComputeInstanceProfileInfo_v2_t::sharedOfaCount

Shared OFA Engine count.

char nvmlComputeInstanceProfileInfo_v2_t::name

Profile name.

5.10. nvmlComputeInstanceProfileInfo_v3_t Struct Reference

Compute instance profile information (v3).

Version 3 adds the nvmlComputeInstanceProfileInfo_v3_t::capabilities field nvmlComputeInstanceProfileInfo_t.

unsigned int nvmlComputeInstanceProfileInfo_v3_t::version

Structure version identifier (set to nvmlComputeInstanceProfileInfo_v3).

unsigned int nvmlComputeInstanceProfileInfo_v3_t::id

Unique profile ID within the GPU instance.

unsigned int nvmlComputeInstanceProfileInfo_v3_t::sliceCount

GPU Slice count.

unsigned int nvmlComputeInstanceProfileInfo_v3_t::instanceCount

Compute instance count.

unsigned int nvmlComputeInstanceProfileInfo_v3_t::multiprocessorCount

Streaming Multiprocessor count.

unsigned int nvmlComputeInstanceProfileInfo_v3_t::sharedCopyEngineCount

Shared Copy Engine count.

unsigned int nvmlComputeInstanceProfileInfo_v3_t::sharedDecoderCount

Shared Decoder Engine count.

unsigned int nvmlComputeInstanceProfileInfo_v3_t::sharedEncoderCount

Shared Encoder Engine count.

unsigned int nvmlComputeInstanceProfileInfo_v3_t::sharedJpegCount

Shared JPEG Engine count.

unsigned int nvmlComputeInstanceProfileInfo_v3_t::sharedOfaCount

Shared OFA Engine count.

char nvmlComputeInstanceProfileInfo_v3_t::name

Profile name.

unsigned int nvmlComputeInstanceProfileInfo_v3_t::capabilities

Additional capabilities.

5.11. nvmlConfComputeMemSizeInfo_t Struct Reference

Protected memory size

5.12. nvmlEccErrorCounts_t Struct Reference

Detailed ECC error counts for a device.

Deprecated Different GPU families can have different memory error counters See nvmlDeviceGetMemoryErrorCounter

unsigned long long nvmlEccErrorCounts_t::l1Cache

L1 cache errors.

unsigned long long nvmlEccErrorCounts_t::l2Cache

L2 cache errors.

unsigned long long nvmlEccErrorCounts_t::deviceMemory

Device memory errors.

unsigned long long nvmlEccErrorCounts_t::registerFile

Register file errors.

5.13. nvmlEccSramErrorStatus_v1_t Struct Reference

Structure to store SRAM uncorrectable error counters

unsigned int nvmlEccSramErrorStatus_v1_t::version

the API version number

unsigned long long nvmlEccSramErrorStatus_v1_t::aggregateUncParity

aggregate uncorrectable parity error count

unsigned long long nvmlEccSramErrorStatus_v1_t::aggregateUncSecDed

aggregate uncorrectable SEC-DED error count

unsigned long long nvmlEccSramErrorStatus_v1_t::aggregateCor

aggregate correctable error count

unsigned long long nvmlEccSramErrorStatus_v1_t::volatileUncParity

volatile uncorrectable parity error count

unsigned long long nvmlEccSramErrorStatus_v1_t::volatileUncSecDed

volatile uncorrectable SEC-DED error count

unsigned long long nvmlEccSramErrorStatus_v1_t::volatileCor

volatile correctable error count

unsigned long long nvmlEccSramErrorStatus_v1_t::aggregateUncBucketL2

aggregate uncorrectable error count for L2 cache bucket

unsigned long long nvmlEccSramErrorStatus_v1_t::aggregateUncBucketSm

aggregate uncorrectable error count for SM bucket

unsigned long long nvmlEccSramErrorStatus_v1_t::aggregateUncBucketPcie

aggregate uncorrectable error count for PCIE bucket

unsigned long long nvmlEccSramErrorStatus_v1_t::aggregateUncBucketMcu

aggregate uncorrectable error count for Microcontroller bucket

unsigned long long nvmlEccSramErrorStatus_v1_t::aggregateUncBucketOther

aggregate uncorrectable error count for Other bucket

unsigned int nvmlEccSramErrorStatus_v1_t::bThresholdExceeded

if the error threshold of field diag is exceeded

5.14. nvmlEncoderSessionInfo_t Struct Reference

Structure to hold encoder session data

unsigned int nvmlEncoderSessionInfo_t::sessionId

Unique session ID.

unsigned int nvmlEncoderSessionInfo_t::pid

Owning process ID.

nvmlVgpuInstance_t nvmlEncoderSessionInfo_t::vgpuInstance

Owning vGPU instance ID (only valid on vGPU hosts, otherwise zero).

nvmlEncoderType_t nvmlEncoderSessionInfo_t::codecType

Video encoder type.

unsigned int nvmlEncoderSessionInfo_t::hResolution

Current encode horizontal resolution.

unsigned int nvmlEncoderSessionInfo_t::vResolution

Current encode vertical resolution.

unsigned int nvmlEncoderSessionInfo_t::averageFps

Moving average encode frames per second.

unsigned int nvmlEncoderSessionInfo_t::averageLatency

Moving average encode latency in microseconds.

5.15. nvmlEventData_t Struct Reference

Information about occurred event

nvmlDevice_t nvmlEventData_t::device
Specific device where the event occurred.

unsigned long long nvmlEventData_t::eventType
Information about what specific event occurred.

unsigned long long nvmlEventData_t::eventData
Stores XID error for the device in the event of nvmlEventTypeXidCriticalError,.

unsigned int nvmlEventData_t::gpuInstanceId
If MIG is enabled and nvmlEventTypeXidCriticalError event is attributable to a GPU.

unsigned int nvmlEventData_t::computeInstanceId
If MIG is enabled and nvmlEventTypeXidCriticalError event is attributable to a.

5.16. nvmlExcludedDeviceInfo_t Struct Reference

Excluded GPU device information

struct nvmlPciInfo_t nvmlExcludedDeviceInfo_t::pciInfo
The PCI information for the excluded GPU.

char nvmlExcludedDeviceInfo_t::uuid
The ASCII string UUID for the excluded GPU.

5.17. nvmlFBCSessionInfo_t Struct Reference

Structure to hold FBC session data

unsigned int nvmlFBCSessionInfo_t::sessionId

Unique session ID.

unsigned int nvmlFBCSessionInfo_t::pid

Owning process ID.

nvmlVgpuInstance_t nvmlFBCSessionInfo_t::vgpuInstance

Owning vGPU instance ID (only valid on vGPU hosts, otherwise zero).

unsigned int nvmlFBCSessionInfo_t::displayOrdinal

Display identifier.

nvmlFBCSessionType_t nvmlFBCSessionInfo_t::sessionType

Type of frame buffer capture session.

unsigned int nvmlFBCSessionInfo_t::sessionFlags

Session flags (one or more of NVML_NVFBC_SESSION_FLAG_XXX).

unsigned int nvmlFBCSessionInfo_t::hMaxResolution

Max horizontal resolution supported by the capture session.

unsigned int nvmlFBCSessionInfo_t::vMaxResolution

Max vertical resolution supported by the capture session.

unsigned int nvmlFBCSessionInfo_t::hResolution

Horizontal resolution requested by caller in capture call.

unsigned int nvmlFBCSessionInfo_t::vResolution

Vertical resolution requested by caller in capture call.

unsigned int nvmlFBCSessionInfo_t::averageFPS

Moving average new frames captured per second.

unsigned int nvmlFBCSessionInfo_t::averageLatency

Moving average new frame capture latency in microseconds.

5.18. nvmlFBCStats_t Struct Reference

Structure to hold frame buffer capture sessions stats

unsigned int nvmlFBCStats_t::sessionsCount

Total no of sessions.

unsigned int nvmlFBCStats_t::averageFPS

Moving average new frames captured per second.

unsigned int nvmlFBCStats_t::averageLatency

Moving average new frame capture latency in microseconds.

5.19. nvmlFieldValue_t Struct Reference

Information for a Field Value Sample

unsigned int nvmlFieldValue_t::fieldId

ID of the NVML field to retrieve. This must be set before any call that uses this struct. See the constants starting with NVML_FI_ above.

unsigned int nvmlFieldValue_t::scopeId

Scope ID can represent data used by NVML depending on fieldId's context. For example, for NVLink throughput counter data, scopeId can represent linkId.

long long nvmlFieldValue_t::timestamp

CPU Timestamp of this value in microseconds since 1970.

long long nvmlFieldValue_t::latencyUsec

How long this field value took to update (in usec) within NVML. This may be averaged across several fields that are serviced by the same driver call.

nvmlValueType_t nvmlFieldValue_t::valueType

Type of the value stored in value.

nvmlReturn_t nvmlFieldValue_t::nvmlReturn

Return code for retrieving this value. This must be checked before looking at value, as value is undefined if nvmlReturn != NVML_SUCCESS.

nvmlFieldValue_t::value

Value for this field. This is only valid if nvmlReturn == NVML_SUCCESS.

5.20. nvmlGpmMetric_t Struct Reference

GPM metric information.

unsigned int nvmlGpmMetric_t::metricId
IN: NVML_GPM_METRIC_? define of which metric to retrieve.

nvmlReturn_t nvmlGpmMetric_t::nvmlReturn
OUT: Status of this metric. If this is nonzero, then value is not valid.

double nvmlGpmMetric_t::value
OUT: Value of this metric. Is only valid if nvmlReturn is 0 (NVML_SUCCESS).

nvmlGpmMetric_t::@6 nvmlGpmMetric_t::metricInfo
OUT: Metric name and unit. Those can be NULL if not defined.

5.21. nvmlGpmMetricsGet_t Struct Reference

GPM buffer information.

unsigned int nvmlGpmMetricsGet_t::version
IN: Set to NVML_GPM_METRICS_GET_VERSION.

unsigned int nvmlGpmMetricsGet_t::numMetrics
IN: How many metrics to retrieve in metrics[].

nvmlGpmSample_t nvmlGpmMetricsGet_t::sample1
IN: Sample buffer.

nvmlGpmSample_t nvmlGpmMetricsGet_t::sample2
IN: Sample buffer.

struct nvmlGpmMetric_t nvmlGpmMetricsGet_t::metrics
IN/OUT: Array of metrics. Set metricId on call. See nvmlReturn and value on return.

5.22. nvmlGpmSupport_t Struct Reference

GPM device information.

unsigned int nvmlGpmSupport_t::version
IN: Set to NVML_GPM_SUPPORT_VERSION.

unsigned int nvmlGpmSupport_t::isSupportedDevice
OUT: Indicates device support.

5.23. nvmlGpuFabricInfo_v2_t Struct Reference

GPU Fabric information (v2).

Version 2 adds the nvmlGpuFabricInfo_v2_t::version field to the start of the structure, and the nvmlGpuFabricInfo_v2_t::healthMask field to the end. This structure is not backwards-compatible with nvmlGpuFabricInfo_t.

unsigned int nvmlGpuFabricInfo_v2_t::version
Structure version identifier (set to nvmlGpuFabricInfo_v2).

unsigned char nvmlGpuFabricInfo_v2_t::clusterUuid
Uuid of the cluster to which this GPU belongs.

nvmlReturn_t nvmlGpuFabricInfo_v2_t::status
Error status, if any. Must be checked only if state returns "complete".

unsigned int nvmlGpuFabricInfo_v2_t::cliqueId
ID of the fabric clique to which this GPU belongs.

nvmlGpuFabricState_t nvmlGpuFabricInfo_v2_t::state
Current state of GPU registration process.

unsigned int nvmlGpuFabricInfo_v2_t::healthMask
GPU Fabric health Status Mask.

5.24. nvmlGpuInstancePlacement_t Struct Reference

MIG compute instance profile capability.

Bit field values representing MIG profile capabilities nvmlComputeInstanceProfileInfo_v3_t::capabilities

unsigned int nvmlGpuInstancePlacement_t::start

Index of first occupied memory slice.

unsigned int nvmlGpuInstancePlacement_t::size

Number of memory slices occupied.

5.25. nvmlGpuInstanceProfileInfo_t Struct Reference

GPU instance profile information.

unsigned int nvmlGpuInstanceProfileInfo_t::id

Unique profile ID within the device.

unsigned int nvmlGpuInstanceProfileInfo_t::isP2pSupported

Peer-to-Peer support.

unsigned int nvmlGpuInstanceProfileInfo_t::sliceCount

GPU Slice count.

unsigned int nvmlGpuInstanceProfileInfo_t::instanceCount

GPU instance count.

unsigned int nvmlGpuInstanceProfileInfo_t::multiprocessorCount

Streaming Multiprocessor count.

unsigned int nvmlGpuInstanceProfileInfo_t::copyEngineCount

Copy Engine count.

unsigned int nvmlGpuInstanceProfileInfo_t::decoderCount

Decoder Engine count.

unsigned int nvmlGpuInstanceProfileInfo_t::encoderCount

Encoder Engine count.

unsigned int nvmlGpuInstanceProfileInfo_t::jpegCount

JPEG Engine count.

unsigned int nvmlGpuInstanceProfileInfo_t::ofaCount

OFA Engine count.

unsigned long long nvmlGpuInstanceProfileInfo_t::memorySizeMB

Memory size in MBytes.

5.26. nvmlGpuInstanceProfileInfo_v2_t Struct Reference

GPU instance profile information (v2).

Version 2 adds the nvmlGpuInstanceProfileInfo_v2_t::version field to the start of the structure, and the nvmlGpuInstanceProfileInfo_v2_t::name field to the end. This structure is not backwards-compatible with nvmlGpuInstanceProfileInfo_t.

unsigned int nvmlGpuInstanceProfileInfo_v2_t::version
Structure version identifier (set to nvmlGpuInstanceProfileInfo_v2).

unsigned int nvmlGpuInstanceProfileInfo_v2_t::id
Unique profile ID within the device.

unsigned int nvmlGpuInstanceProfileInfo_v2_t::isP2pSupported
Peer-to-Peer support.

unsigned int nvmlGpuInstanceProfileInfo_v2_t::sliceCount
GPU Slice count.

unsigned int nvmlGpuInstanceProfileInfo_v2_t::instanceCount
GPU instance count.

unsigned int nvmlGpuInstanceProfileInfo_v2_t::multiprocessorCount
Streaming Multiprocessor count.

unsigned int nvmlGpuInstanceProfileInfo_v2_t::copyEngineCount
Copy Engine count.

unsigned int nvmlGpuInstanceProfileInfo_v2_t::decoderCount
Decoder Engine count.

unsigned int nvmlGpuInstanceProfileInfo_v2_t::encoderCount
Encoder Engine count.

unsigned int nvmlGpuInstanceProfileInfo_v2_t::jpegCount
JPEG Engine count.

unsigned int nvmlGpuInstanceProfileInfo_v2_t::ofaCount

OFA Engine count.

unsigned long long nvmlGpuInstanceProfileInfo_v2_t::memorySizeMB

Memory size in MBytes.

char nvmlGpuInstanceProfileInfo_v2_t::name

Profile name.

5.27. nvmlGpuInstanceProfileInfo_v3_t Struct Reference

GPU instance profile information (v3).

Version 3 removes isP2pSupported field and adds the nvmlGpuInstanceProfileInfo_v3_t::capabilities field nvmlGpuInstanceProfileInfo_t.

unsigned int nvmlGpuInstanceProfileInfo_v3_t::version

Structure version identifier (set to nvmlGpuInstanceProfileInfo_v3).

unsigned int nvmlGpuInstanceProfileInfo_v3_t::id

Unique profile ID within the device.

unsigned int nvmlGpuInstanceProfileInfo_v3_t::sliceCount

GPU Slice count.

unsigned int nvmlGpuInstanceProfileInfo_v3_t::instanceCount

GPU instance count.

unsigned int nvmlGpuInstanceProfileInfo_v3_t::multiprocessorCount

Streaming Multiprocessor count.

unsigned int nvmlGpuInstanceProfileInfo_v3_t::copyEngineCount

Copy Engine count.

unsigned int nvmlGpuInstanceProfileInfo_v3_t::decoderCount

Decoder Engine count.

unsigned int nvmlGpuInstanceProfileInfo_v3_t::encoderCount

Encoder Engine count.

unsigned int nvmlGpuInstanceProfileInfo_v3_t::jpegCount

JPEG Engine count.

unsigned int nvmlGpuInstanceProfileInfo_v3_t::ofaCount

OFA Engine count.

unsigned long long nvmlGpuInstanceProfileInfo_v3_t::memorySizeMB

Memory size in MBytes.

char nvmlGpuInstanceProfileInfo_v3_t::name

Profile name.

unsigned int nvmlGpuInstanceProfileInfo_v3_t::capabilities

Additional capabilities.

5.28. nvmlGridLicensableFeature_t Struct Reference

Structure containing vGPU software licensable feature information

nvmlGridLicenseFeatureCode_t nvmlGridLicensableFeature_t::featureCode

Licensed feature code.

unsigned int nvmlGridLicensableFeature_t::featureState

Non-zero if feature is currently licensed, otherwise zero.

char nvmlGridLicensableFeature_t::licenseInfo

Deprecated.

char nvmlGridLicensableFeature_t::productName

Product name of feature.

unsigned int nvmlGridLicensableFeature_t::featureEnabled

Non-zero if feature is enabled, otherwise zero.

struct nvmlGridLicenseExpiry_t nvmlGridLicensableFeature_t::licenseExpiry

License expiry structure containing date and time.

5.29. nvmlGridLicensableFeatures_t Struct Reference

Structure to store vGPU software licensable features

int nvmlGridLicensableFeatures_t::isGridLicenseSupported

Non-zero if vGPU Software Licensing is supported on the system, otherwise zero.

unsigned int nvmlGridLicensableFeatures_t::licensableFeaturesCount

Entries returned in gridLicensableFeatures array.

struct nvmlGridLicensableFeature_t nvmlGridLicensableFeatures_t::gridLicensableFeatures

Array of vGPU software licensable features.

5.30. nvmlGridLicenseExpiry_t Struct Reference

Structure to store license expiry date and time values

unsigned int nvmlGridLicenseExpiry_t::year
Year value of license expiry.

unsigned short nvmlGridLicenseExpiry_t::month
Month value of license expiry.

unsigned short nvmlGridLicenseExpiry_t::day
Day value of license expiry.

unsigned short nvmlGridLicenseExpiry_t::hour
Hour value of license expiry.

unsigned short nvmlGridLicenseExpiry_t::min
Minutes value of license expiry.

unsigned short nvmlGridLicenseExpiry_t::sec
Seconds value of license expiry.

unsigned char nvmlGridLicenseExpiry_t::status
License expiry status.

5.31. nvmlHwbcEntry_t Struct Reference

Description of HWBC entry

5.32. nvmlLedState_t Struct Reference

LED states for an S-class unit.

char nvmlLedState_t::cause

If amber, a text description of the cause.

nvmlLedColor_t nvmlLedState_t::color

GREEN or AMBER.

5.33. nvmlMemory_t Struct Reference

Memory allocation information for a device (v1). The total amount is equal to the sum of the amounts of free and used memory.

unsigned long long nvmlMemory_t::total

Total physical device memory (in bytes).

unsigned long long nvmlMemory_t::free

Unallocated device memory (in bytes).

unsigned long long nvmlMemory_t::used

Description

Sum of Reserved and Allocated device memory (in bytes). Note that the driver/GPU always sets aside a small amount of memory for bookkeeping

5.34. nvmlMemory_v2_t Struct Reference

Memory allocation information for a device (v2).

Version 2 adds versioning for the struct and the amount of system-reserved memory as an output.

unsigned int nvmlMemory_v2_t::version

Structure format version (must be 2).

unsigned long long nvmlMemory_v2_t::total

Total physical device memory (in bytes).

unsigned long long nvmlMemory_v2_t::reserved

Device memory (in bytes) reserved for system use (driver or firmware).

unsigned long long nvmlMemory_v2_t::free

Unallocated device memory (in bytes).

unsigned long long nvmlMemory_v2_t::used

Allocated device memory (in bytes).

5.35. nvmlNvLinkUtilizationControl_t Struct Reference

Struct to define the NVLINK counter controls

5.36. nvmlPciInfo_t Struct Reference

PCI information about a GPU device.

char nvmlPciInfo_t::busIdLegacy

The legacy tuple domain:bus:device.function PCI identifier (& NULL terminator).

unsigned int nvmlPciInfo_t::domain

The PCI domain on which the device's bus resides, 0 to 0xffffffff.

unsigned int nvmlPciInfo_t::bus

The bus on which the device resides, 0 to 0xff.

unsigned int nvmlPciInfo_t::device

The device's id on the bus, 0 to 31.

unsigned int nvmlPciInfo_t::pciDeviceId

The combined 16-bit device id and 16-bit vendor id.

unsigned int nvmlPciInfo_t::pciSubSystemId

The 32-bit Sub System Device ID.

char nvmlPciInfo_t::busId

The tuple domain:bus:device.function PCI identifier (& NULL terminator).

5.37. nvmlPciInfoExt_v1_t Struct Reference

PCI information about a GPU device.

unsigned int nvmlPciInfoExt_v1_t::version
The version number of this struct.

unsigned int nvmlPciInfoExt_v1_t::domain
The PCI domain on which the device's bus resides, 0 to 0xffffffff.

unsigned int nvmlPciInfoExt_v1_t::bus
The bus on which the device resides, 0 to 0xff.

unsigned int nvmlPciInfoExt_v1_t::device
The device's id on the bus, 0 to 31.

unsigned int nvmlPciInfoExt_v1_t::pciDeviceId
The combined 16-bit device id and 16-bit vendor id.

unsigned int nvmlPciInfoExt_v1_t::pciSubSystemId
The 32-bit Sub System Device ID.

unsigned int nvmlPciInfoExt_v1_t::baseClass
The 8-bit PCI base class code.

unsigned int nvmlPciInfoExt_v1_t::subClass
The 8-bit PCI sub class code.

char nvmlPciInfoExt_v1_t::busId
The tuple domain:bus:device.function PCI identifier (& NULL terminator).

5.38. nvmlProcessDetail_v1_t Struct Reference

Information about running process on the GPU with protected memory

unsigned int nvmlProcessDetail_v1_t::pid

Process ID.

unsigned long long nvmlProcessDetail_v1_t::usedGpuMemory

Description

Amount of used GPU memory in bytes. Under WDDM, NVML_VALUE_NOT_AVAILABLE is always reported because Windows KMD manages all the memory and not the NVIDIA driver

unsigned int nvmlProcessDetail_v1_t::gpuInstanceId

If MIG is enabled, stores a valid GPU instance ID. gpuInstanceId is.

unsigned int nvmlProcessDetail_v1_t::computeInstanceId

If MIG is enabled, stores a valid compute instance ID. computeInstanceId.

unsigned long long nvmlProcessDetail_v1_t::usedGpuCcProtectedMemory

Amount of used GPU conf compute protected memory in bytes.

5.39. nvmlProcessDetailList_v1_t Struct Reference

Information about all running processes on the GPU for the given mode

unsigned int nvmlProcessDetailList_v1_t::version

Struct version, MUST be nvmlProcessDetailList_v1.

unsigned int nvmlProcessDetailList_v1_t::mode

Process mode(Compute/Graphics/MPSCompute).

unsigned int nvmlProcessDetailList_v1_t::numProcArrayEntries

Number of process entries in procArray.

nvmlProcessDetail_v1_t *nvmlProcessDetailList_v1_t::procArray

Process array.

5.40. nvmlProcessesUtilizationInfo_v1_t Struct Reference

Structure to store utilization and process ID for each running process -- version 1

unsigned int nvmlProcessesUtilizationInfo_v1_t::version

The version number of this struct.

unsigned int nvmlProcessesUtilizationInfo_v1_t::processSamplesCount

Caller-supplied array size, and returns number of processes running.

unsigned long long nvmlProcessesUtilizationInfo_v1_t::lastSeenTimeStamp

Return only samples with timestamp greater than lastSeenTimeStamp.

nvmlProcessUtilizationInfo_v1_t *nvmlProcessesUtilizationInfo_v1_t::procUtilArray

The array (allocated by caller) of the utilization of GPU SM, framebuffer, video encoder, video decoder, JPEG, and OFA.

5.41. nvmlProcessInfo_t Struct Reference

Information about running compute processes on the GPU

unsigned int nvmlProcessInfo_t::pid

Process ID.

unsigned long long nvmlProcessInfo_t::usedGpuMemory

Description

Amount of used GPU memory in bytes. Under WDDM, NVML_VALUE_NOT_AVAILABLE is always reported because Windows KMD manages all the memory and not the NVIDIA driver

unsigned int nvmlProcessInfo_t::gpuInstanceId

If MIG is enabled, stores a valid GPU instance ID. gpuInstanceId is set to.

unsigned int nvmlProcessInfo_t::computeInstanceId

If MIG is enabled, stores a valid compute instance ID. computeInstanceId is set to.

5.42. nvmlProcessInfo_v1_t Struct Reference

Information about running compute processes on the GPU, legacy version for older versions of the API.

unsigned int nvmlProcessInfo_v1_t::pid

Process ID.

unsigned long long nvmlProcessInfo_v1_t::usedGpuMemory

Description

Amount of used GPU memory in bytes. Under WDDM, NVML_VALUE_NOT_AVAILABLE is always reported because Windows KMD manages all the memory and not the NVIDIA driver

5.43. nvmlProcessUtilizationInfo_v1_t Struct Reference

Structure to store utilization value and process Id -- version 1

unsigned long long nvmlProcessUtilizationInfo_v1_t::timeStamp

CPU Timestamp in microseconds.

unsigned int nvmlProcessUtilizationInfo_v1_t::pid

PID of process.

unsigned int nvmlProcessUtilizationInfo_v1_t::smUtil

SM (3D/Compute) Util Value.

unsigned int nvmlProcessUtilizationInfo_v1_t::memUtil

Frame Buffer Memory Util Value.

unsigned int nvmlProcessUtilizationInfo_v1_t::encUtil

Encoder Util Value.

unsigned int nvmlProcessUtilizationInfo_v1_t::decUtil

Decoder Util Value.

unsigned int nvmlProcessUtilizationInfo_v1_t::jpgUtil

Jpeg Util Value.

unsigned int nvmlProcessUtilizationInfo_v1_t::ofaUtil

Ofa Util Value.

5.44. nvmlProcessUtilizationSample_t Struct Reference

Structure to store utilization value and process Id

unsigned int nvmlProcessUtilizationSample_t::pid

PID of process.

unsigned long long nvmlProcessUtilizationSample_t::timeStamp

CPU Timestamp in microseconds.

unsigned int nvmlProcessUtilizationSample_t::smUtil

SM (3D/Compute) Util Value.

unsigned int nvmlProcessUtilizationSample_t::memUtil

Frame Buffer Memory Util Value.

unsigned int nvmlProcessUtilizationSample_t::encUtil

Encoder Util Value.

unsigned int nvmlProcessUtilizationSample_t::decUtil

Decoder Util Value.

5.45. nvmlPSUInfo_t Struct Reference

Power usage information for an S-class unit. The power supply state is a human readable string that equals "Normal" or contains a combination of "Abnormal" plus one or more of the following:

- High voltage
- Fan failure
- Heatsink temperature
- Current limit
- Voltage below UV alarm threshold
- Low-voltage
- SI2C remote off command
- MOD_DISABLE input
- Short pin transition

char nvmlPSUInfo_t::state

The power supply state.

unsigned int nvmlPSUInfo_t::current

PSU current (A).

unsigned int nvmlPSUInfo_t::voltage

PSU voltage (V).

unsigned int nvmlPSUInfo_t::power

PSU power draw (W).

5.46. nvmlRowRemapperHistogramValues_t Struct Reference

Possible values that classify the remap availability for each bank. The max field will contain the number of banks that have maximum remap availability (all reserved rows are available). None means that there are no reserved rows available.

5.47. nvmlSample_t Struct Reference

Information for Sample

unsigned long long nvmlSample_t::timeStamp

CPU Timestamp in microseconds.

nvmlSample_t::sampleValue

Sample Value.

5.48. nvmlSystemConfComputeSettings_v1_t Struct Reference

Confidential Compute System settings

5.49. nvmlUnitFanInfo_t Struct Reference

Fan speed reading for a single fan in an S-class unit.

unsigned int nvmlUnitFanInfo_t::speed

Fan speed (RPM).

nvmlFanState_t nvmlUnitFanInfo_t::state

Flag that indicates whether fan is working properly.

5.50. nvmlUnitFanSpeeds_t Struct Reference

Fan speed readings for an entire S-class unit.

struct nvmlUnitFanInfo_t nvmlUnitFanSpeeds_t::fans

Fan speed data for each fan.

unsigned int nvmlUnitFanSpeeds_t::count

Number of fans in unit.

5.51. nvmlUnitInfo_t Struct Reference

Static S-class unit info.

char nvmlUnitInfo_t::name

Product name.

char nvmlUnitInfo_t::id

Product identifier.

char nvmlUnitInfo_t::serial

Product serial number.

char nvmlUnitInfo_t::firmwareVersion

Firmware version.

5.52. nvmlUtilization_t Struct Reference

Utilization information for a device. Each sample period may be between 1 second and 1/6 second, depending on the product being queried.

unsigned int nvmlUtilization_t::gpu

Percent of time over the past sample period during which one or more kernels was executing on the GPU.

unsigned int nvmlUtilization_t::memory

Percent of time over the past sample period during which global (device) memory was being read or written.

5.53. nvmlValue_t Union Reference

Union to represent different types of Value

double nvmlValue_t::dVal

If the value is double.

int nvmlValue_t::siVal

If the value is signed int.

unsigned int nvmlValue_t::uiVal

If the value is unsigned int.

unsignedlong nvmlValue_t::ulVal

If the value is unsigned long.

unsigned long long nvmlValue_t::ullVal

If the value is unsigned long long.

signed long long nvmlValue_t::sllVal

If the value is signed long long.

5.54. nvmlVgpuHeterogeneousMode_v1_t Struct Reference

Structure to store the vGPU heterogeneous mode of device -- version 1

unsigned int nvmlVgpuHeterogeneousMode_v1_t::version

The version number of this struct.

unsigned int nvmlVgpuHeterogeneousMode_v1_t::mode

The vGPU heterogeneous mode.

5.55. nvmlVgpuInstancesUtilizationInfo_v1_t Struct Reference

Structure to store recent utilization for vGPU instances running on a device -- version 1

unsigned int nvmlVgpuInstancesUtilizationInfo_v1_t::version

The version number of this struct.

nvmlValueType_t nvmlVgpuInstancesUtilizationInfo_v1_t::sampleValType

Hold the type of returned sample values.

unsigned int nvmlVgpuInstancesUtilizationInfo_v1_t::vgpuInstanceCount

Hold the number of vGPU instances.

unsigned long long nvmlVgpuInstancesUtilizationInfo_v1_t::lastSeenTimeStamp

Return only samples with timestamp greater than lastSeenTimeStamp.

nvmlVgpuInstanceUtilizationInfo_v1_t *nvmlVgpuInstancesUtilizationInfo_v1_t::vgpuUtilArray

The array (allocated by caller) in which vGPU utilization are returned.

5.56. nvmlVgpuInstanceUtilizationInfo_v1_t Struct Reference

Structure to store Utilization Value and vgpuInstance Info -- Version 1

unsigned long long nvmlVgpuInstanceUtilizationInfo_v1_t::timeStamp

CPU Timestamp in microseconds.

nvmlVgpuInstance_t nvmlVgpuInstanceUtilizationInfo_v1_t::vgpuInstance

vGPU Instance

nvmlVgpuInstanceUtilizationInfo_v1_t::smUtil

SM (3D/Compute) Util Value.

nvmlVgpuInstanceUtilizationInfo_v1_t::memUtil

Frame Buffer Memory Util Value.

nvmlVgpuInstanceUtilizationInfo_v1_t::encUtil

Encoder Util Value.

nvmlVgpuInstanceUtilizationInfo_v1_t::decUtil

Decoder Util Value.

nvmlVgpuInstanceUtilizationInfo_v1_t::jpgUtil

Jpeg Util Value.

nvmlVgpuInstanceUtilizationInfo_v1_t::ofaUtil

Ofa Util Value.

5.57. nvmlVgpuInstanceUtilizationSample_t Struct Reference

Structure to store Utilization Value and vgpuInstance

nvmlVgpuInstance_t nvmlVgpuInstanceUtilizationSample_t::vgpuInstance

vGPU Instance

unsigned long long nvmlVgpuInstanceUtilizationSample_t::timeStamp

CPU Timestamp in microseconds.

nvmlVgpuInstanceUtilizationSample_t::smUtil

SM (3D/Compute) Util Value.

nvmlVgpuInstanceUtilizationSample_t::memUtil

Frame Buffer Memory Util Value.

nvmlVgpuInstanceUtilizationSample_t::encUtil

Encoder Util Value.

nvmlVgpuInstanceUtilizationSample_t::decUtil

Decoder Util Value.

5.58. nvmlVgpuLicenseExpiry_t Struct Reference

Structure to store the vGPU license expiry details

unsigned int nvmlVgpuLicenseExpiry_t::year
Year of license expiry.

unsigned short nvmlVgpuLicenseExpiry_t::month
Month of license expiry.

unsigned short nvmlVgpuLicenseExpiry_t::day
Day of license expiry.

unsigned short nvmlVgpuLicenseExpiry_t::hour
Hour of license expiry.

unsigned short nvmlVgpuLicenseExpiry_t::min
Minutes of license expiry.

unsigned short nvmlVgpuLicenseExpiry_t::sec
Seconds of license expiry.

unsigned char nvmlVgpuLicenseExpiry_t::status
License expiry status.

5.59. nvmlVgpuMetadata_t Struct Reference

vGPU metadata structure.

unsigned int nvmlVgpuMetadata_t::version
Current version of the structure.

unsigned int nvmlVgpuMetadata_t::revision
Current revision of the structure.

nvmlVgpuGuestInfoState_t nvmlVgpuMetadata_t::guestInfoState
Current state of Guest-dependent fields.

char nvmlVgpuMetadata_t::guestDriverVersion
Version of driver installed in guest.

char nvmlVgpuMetadata_t::hostDriverVersion
Version of driver installed in host.

unsigned int nvmlVgpuMetadata_t::reserved
Reserved for internal use.

unsigned int nvmlVgpuMetadata_t::vgpuVirtualizationCaps
vGPU virtualization capabilities bitfield

unsigned int nvmlVgpuMetadata_t::guestVgpuVersion
vGPU version of guest driver

unsigned int nvmlVgpuMetadata_t::opaqueDataSize
Size of opaque data field in bytes.

char nvmlVgpuMetadata_t::opaqueData
Opaque data.

5.60. nvmlVgpuPgpuCompatibility_t Struct Reference

vGPU-pGPU compatibility structure

nvmlVgpuVmCompatibility_t nvmlVgpuPgpuCompatibility_t::vgpuVmCompatibility

Compatibility of vGPU VM. See nvmlVgpuVmCompatibility_t.

nvmlVgpuPgpuCompatibilityLimitCode_t nvmlVgpuPgpuCompatibility_t::compatibilityLimitCode

Limiting factor for vGPU-pGPU compatibility. See nvmlVgpuPgpuCompatibilityLimitCode_t.

5.61. nvmlVgpuPgpuMetadata_t Struct Reference

Physical GPU metadata structure

unsigned int nvmlVgpuPgpuMetadata_t::version
Current version of the structure.

unsigned int nvmlVgpuPgpuMetadata_t::revision
Current revision of the structure.

char nvmlVgpuPgpuMetadata_t::hostDriverVersion
Host driver version.

unsigned int nvmlVgpuPgpuMetadata_t::pgpuVirtualizationCaps
Pgpu virtualization capabilities bitfield.

unsigned int nvmlVgpuPgpuMetadata_t::reserved
Reserved for internal use.

struct nvmlVgpuVersion_t nvmlVgpuPgpuMetadata_t::hostSupportedVgpuRange
vGPU version range supported by host driver

unsigned int nvmlVgpuPgpuMetadata_t::opaqueDataSize
Size of opaque data field in bytes.

char nvmlVgpuPgpuMetadata_t::opaqueData
Opaque data.

5.62. nvmlVgpuPlacementId_v1_t Struct Reference

Structure to store the placement ID of vGPU instance -- version 1

unsigned int nvmlVgpuPlacementId_v1_t::version

The version number of this struct.

unsigned int nvmlVgpuPlacementId_v1_t::placementId

Placement ID of the active vGPU instance.

5.63. nvmlVgpuPlacementList_v1_t Struct Reference

Structure to store the list of vGPU placements -- version 1

unsigned int nvmlVgpuPlacementList_v1_t::version

The version number of this struct.

unsigned int nvmlVgpuPlacementList_v1_t::placementSize

The number of slots occupied by the vGPU type.

unsigned int nvmlVgpuPlacementList_v1_t::count

Count of placement IDs fetched.

unsigned int *nvmlVgpuPlacementList_v1_t::placementIds

Placement IDs for the vGPU type.

5.64. nvmlVgpuProcessesUtilizationInfo_v1_t Struct Reference

Structure to store recent utilization, vgpuInstance and subprocess information for processes running on vGPU instances active on a device -- version 1

unsigned int nvmlVgpuProcessesUtilizationInfo_v1_t::version

The version number of this struct.

unsigned int nvmlVgpuProcessesUtilizationInfo_v1_t::vgpuProcessCount

Hold the number of processes running on vGPU instances.

unsigned long long nvmlVgpuProcessesUtilizationInfo_v1_t::lastSeenTimeStamp

Return only samples with timestamp greater than lastSeenTimeStamp.

nvmlVgpuProcessUtilizationInfo_v1_t *nvmlVgpuProcessesUtilizationInfo_v1_t::vgpuProcUtilArray

The array (allocated by caller) in which utilization of processes running on vGPU instances are returned.

5.65. nvmlVgpuProcessUtilizationInfo_v1_t Struct Reference

Structure to store Utilization Value, vgpuInstance and subprocess information for process running on vGPU instance -- version 1

char nvmlVgpuProcessUtilizationInfo_v1_t::processName

Name of process running within the vGPU VM.

unsigned long long nvmlVgpuProcessUtilizationInfo_v1_t::timeStamp

CPU Timestamp in microseconds.

nvmlVgpuInstance_t nvmlVgpuProcessUtilizationInfo_v1_t::vgpuInstance

vGPU Instance

unsigned int nvmlVgpuProcessUtilizationInfo_v1_t::pid

PID of process running within the vGPU VM.

unsigned int nvmlVgpuProcessUtilizationInfo_v1_t::smUtil

SM (3D/Compute) Util Value.

unsigned int nvmlVgpuProcessUtilizationInfo_v1_t::memUtil

Frame Buffer Memory Util Value.

unsigned int nvmlVgpuProcessUtilizationInfo_v1_t::encUtil

Encoder Util Value.

unsigned int nvmlVgpuProcessUtilizationInfo_v1_t::decUtil

Decoder Util Value.

unsigned int nvmlVgpuProcessUtilizationInfo_v1_t::jpgUtil

Jpeg Util Value.

unsigned int nvmlVgpuProcessUtilizationInfo_v1_t::ofaUtil

Ofa Util Value.

5.66. nvmlVgpuProcessUtilizationSample_t Struct Reference

Structure to store Utilization Value, vgpuInstance and subprocess information

nvmlVgpuInstance_t nvmlVgpuProcessUtilizationSample_t::vgpuInstance

vGPU Instance

unsigned int nvmlVgpuProcessUtilizationSample_t::pid

PID of process running within the vGPU VM.

char nvmlVgpuProcessUtilizationSample_t::processName

Name of process running within the vGPU VM.

unsigned long long nvmlVgpuProcessUtilizationSample_t::timeStamp

CPU Timestamp in microseconds.

unsigned int nvmlVgpuProcessUtilizationSample_t::smUtil

SM (3D/Compute) Util Value.

unsigned int nvmlVgpuProcessUtilizationSample_t::memUtil

Frame Buffer Memory Util Value.

unsigned int nvmlVgpuProcessUtilizationSample_t::encUtil

Encoder Util Value.

unsigned int nvmlVgpuProcessUtilizationSample_t::decUtil

Decoder Util Value.

5.67. nvmlVgpuSchedulerCapabilities_t Struct Reference

Structure to store the vGPU scheduler capabilities

unsigned int nvmlVgpuSchedulerCapabilities_t::supportedSchedulers

List the supported vGPU schedulers on the device.

unsigned int nvmlVgpuSchedulerCapabilities_t::maxTimeslice

Maximum timeslice value in ns.

unsigned int nvmlVgpuSchedulerCapabilities_t::minTimeslice

Minimum timeslice value in ns.

unsigned int nvmlVgpuSchedulerCapabilities_t::isArrModeSupported

Flag to check Adaptive Round Robin mode enabled/disabled.

unsigned int nvmlVgpuSchedulerCapabilities_t::maxFrequencyForARR

Maximum frequency for Adaptive Round Robin mode.

unsigned int nvmlVgpuSchedulerCapabilities_t::minFrequencyForARR

Minimum frequency for Adaptive Round Robin mode.

unsigned int nvmlVgpuSchedulerCapabilities_t::maxAvgFactorForARR

Maximum averaging factor for Adaptive Round Robin mode.

unsigned int nvmlVgpuSchedulerCapabilities_t::minAvgFactorForARR

Minimum averaging factor for Adaptive Round Robin mode.

5.68. nvmlVgpuSchedulerGetState_t Struct Reference

Structure to store the vGPU scheduler state

unsigned int nvmlVgpuSchedulerGetState_t::schedulerPolicy

Scheduler policy.

unsigned int nvmlVgpuSchedulerGetState_t::arrMode

Adaptive Round Robin scheduler mode. One of the NVML_VGPU_SCHEDULER_ARR_*.

5.69. nvmlVgpuSchedulerLog_t Struct Reference

Structure to store a vGPU software scheduler log

unsigned int nvmlVgpuSchedulerLog_t::engineId

Engine whose software runlist log entries are fetched.

unsigned int nvmlVgpuSchedulerLog_t::schedulerPolicy

Scheduler policy.

unsigned int nvmlVgpuSchedulerLog_t::arrMode

Adaptive Round Robin scheduler mode. One of the NVML_VGPU_SCHEDULER_ARR_*.

unsigned int nvmlVgpuSchedulerLog_t::entriesCount

Count of log entries fetched.

5.70. nvmlVgpuSchedulerLogEntry_t Struct Reference

Structure to store the state and logs of a software runlist

unsigned long long nvmlVgpuSchedulerLogEntry_t::timestamp

Timestamp in ns when this software runlist was preeempted.

unsigned long long nvmlVgpuSchedulerLogEntry_t::timeRunTotal

Total time in ns this software runlist has run.

unsigned long long nvmlVgpuSchedulerLogEntry_t::timeRun

Time in ns this software runlist ran before preemption.

unsigned int nvmlVgpuSchedulerLogEntry_t::swRunlistId

Software runlist Id.

unsigned long long nvmlVgpuSchedulerLogEntry_t::targetTimeSlice

The actual timeslice after deduction.

unsigned long long nvmlVgpuSchedulerLogEntry_t::cumulativePreemptionTime

Preemption time in ns for this SW runlist.

5.71. nvmlVgpuSchedulerParams_t Union Reference

Union to represent the vGPU Scheduler Parameters

unsigned int nvmlVgpuSchedulerParams_t::avgFactor

Average factor in compensating the timeslice for Adaptive Round Robin mode.

unsigned int nvmlVgpuSchedulerParams_t::timeslice

The timeslice in ns for each software run list as configured, or the default value otherwise.

5.72. nvmlVgpuSchedulerSetParams_t Union Reference

Union to represent the vGPU Scheduler set Parameters

unsigned int nvmlVgpuSchedulerSetParams_t::avgFactor

Average factor in compensating the timeslice for Adaptive Round Robin mode.

unsigned int nvmlVgpuSchedulerSetParams_t::frequency

Frequency for Adaptive Round Robin mode.

unsigned int nvmlVgpuSchedulerSetParams_t::timeslice

The timeslice in ns(Nanoseconds) for each software run list as configured, or the default value otherwise.

5.73. nvmlVgpuSchedulerSetState_t Struct Reference

Structure to set the vGPU scheduler state

unsigned int nvmlVgpuSchedulerSetState_t::schedulerPolicy

Scheduler policy.

unsigned int nvmlVgpuSchedulerSetState_t::enableARRMode

Adaptive Round Robin scheduler.

5.74. nvmlVgpuVersion_t Struct Reference

Structure representing range of vGPU versions.

unsigned int nvmlVgpuVersion_t::minVersion

Minimum vGPU version.

unsigned int nvmlVgpuVersion_t::maxVersion

Maximum vGPU version.

5.75. nvmlViolationTime_t Struct Reference

Struct to hold perf policy violation status data

unsigned long long nvmlViolationTime_t::referenceTime

referenceTime represents CPU timestamp in microseconds

unsigned long long nvmlViolationTime_t::violationTime

violationTime in Nanoseconds

Chapter 6. DATA FIELDS

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

A

aggregateCor
- nvmlEccSramErrorStatus_v1_t

aggregateUncBucketL2
- nvmlEccSramErrorStatus_v1_t

aggregateUncBucketMcu
- nvmlEccSramErrorStatus_v1_t

aggregateUncBucketOther
- nvmlEccSramErrorStatus_v1_t

aggregateUncBucketPcie
- nvmlEccSramErrorStatus_v1_t

aggregateUncBucketSm
- nvmlEccSramErrorStatus_v1_t

aggregateUncParity
- nvmlEccSramErrorStatus_v1_t

aggregateUncSecDed
- nvmlEccSramErrorStatus_v1_t

arrMode
- nvmlVgpuSchedulerLog_t
- nvmlVgpuSchedulerGetState_t

averageFps
- nvmlEncoderSessionInfo_t

averageFPS
- nvmlFBCSessionInfo_t
- nvmlFBCStats_t

averageLatency
- nvmlFBCStats_t

memUtil
- nvmlProcessUtilizationInfo_v1_t
- nvmlVgpuInstanceUtilizationSample_t
- nvmlVgpuInstanceUtilizationInfo_v1_t
- nvmlVgpuProcessUtilizationSample_t
- nvmlVgpuProcessUtilizationInfo_v1_t
- nvmlProcessUtilizationSample_t

metricId
- nvmlGpmMetric_t

metricInfo
- nvmlGpmMetric_t

metrics
- nvmlGpmMetricsGet_t

min
- nvmlVgpuLicenseExpiry_t
- nvmlGridLicenseExpiry_t

minAvgFactorForARR
- nvmlVgpuSchedulerCapabilities_t

minFrequencyForARR
- nvmlVgpuSchedulerCapabilities_t

minTimeslice
- nvmlVgpuSchedulerCapabilities_t

minVersion
- nvmlVgpuVersion_t

mode
- nvmlVgpuHeterogeneousMode_v1_t
- nvmlProcessDetailList_v1_t

month
- nvmlGridLicenseExpiry_t
- nvmlVgpuLicenseExpiry_t

multiprocessorCount
- nvmlGpuInstanceProfileInfo_v3_t
- nvmlGpuInstanceProfileInfo_v2_t
- nvmlComputeInstanceProfileInfo_v3_t
- nvmlGpuInstanceProfileInfo_t
- nvmlComputeInstanceProfileInfo_v2_t
- nvmlComputeInstanceProfileInfo_t

N

name
- nvmlUnitInfo_t
- nvmlGpuInstanceProfileInfo_v2_t
- nvmlComputeInstanceProfileInfo_v2_t

nvmlFBCSessionInfo_t
nvmlEncoderSessionInfo_t
nvmlProcessUtilizationInfo_v1_t
nvmlProcessInfo_v1_t

placementId
nvmlVgpuPlacementId_v1_t

placementIds
nvmlVgpuPlacementList_v1_t

placementSize
nvmlVgpuPlacementList_v1_t

power
nvmlPSUInfo_t

procArray
nvmlProcessDetailList_v1_t

processName
nvmlVgpuProcessUtilizationInfo_v1_t
nvmlVgpuProcessUtilizationSample_t

processSamplesCount
nvmlProcessesUtilizationInfo_v1_t

procUtilArray
nvmlProcessesUtilizationInfo_v1_t

productName
nvmlGridLicensableFeature_t

R

referenceTime
nvmlViolationTime_t

registerFile
nvmlEccErrorCounts_t

reserved
nvmlAccountingStats_t
nvmlVgpuPgpuMetadata_t
nvmlVgpuMetadata_t
nvmlMemory_v2_t

revision
nvmlVgpuPgpuMetadata_t
nvmlVgpuMetadata_t

S

sample1
nvmlGpmMetricsGet_t

sample2
nvmlGpmMetricsGet_t

sampleValType
- nvmlVgpuInstancesUtilizationInfo_v1_t

sampleValue
- nvmlSample_t

schedulerPolicy
- nvmlVgpuSchedulerLog_t
- nvmlVgpuSchedulerGetState_t
- nvmlVgpuSchedulerSetState_t

scopeId
- nvmlFieldValue_t

sec
- nvmlVgpuLicenseExpiry_t
- nvmlGridLicenseExpiry_t

serial
- nvmlUnitInfo_t

sessionFlags
- nvmlFBCSessionInfo_t

sessionId
- nvmlEncoderSessionInfo_t
- nvmlFBCSessionInfo_t

sessionsCount
- nvmlFBCStats_t

sessionType
- nvmlFBCSessionInfo_t

sharedCopyEngineCount
- nvmlComputeInstanceProfileInfo_t
- nvmlComputeInstanceProfileInfo_v2_t
- nvmlComputeInstanceProfileInfo_v3_t

sharedDecoderCount
- nvmlComputeInstanceProfileInfo_t
- nvmlComputeInstanceProfileInfo_v2_t
- nvmlComputeInstanceProfileInfo_v3_t

sharedEncoderCount
- nvmlComputeInstanceProfileInfo_t
- nvmlComputeInstanceProfileInfo_v2_t
- nvmlComputeInstanceProfileInfo_v3_t

sharedJpegCount
- nvmlComputeInstanceProfileInfo_t
- nvmlComputeInstanceProfileInfo_v2_t
- nvmlComputeInstanceProfileInfo_v3_t

sharedOfaCount
- nvmlComputeInstanceProfileInfo_t
- nvmlComputeInstanceProfileInfo_v2_t

nvmlComputeInstanceProfileInfo_v3_t

siVal

nvmlValue_t

size

nvmlGpuInstancePlacement_t

sliceCount

nvmlGpuInstanceProfileInfo_t
nvmlGpuInstanceProfileInfo_v2_t
nvmlGpuInstanceProfileInfo_v3_t
nvmlComputeInstanceProfileInfo_t
nvmlComputeInstanceProfileInfo_v2_t
nvmlComputeInstanceProfileInfo_v3_t

sllVal

nvmlValue_t

smUtil

nvmlVgpuInstanceUtilizationInfo_v1_t
nvmlVgpuInstanceUtilizationSample_t
nvmlVgpuProcessUtilizationInfo_v1_t
nvmlProcessUtilizationSample_t
nvmlVgpuProcessUtilizationSample_t
nvmlProcessUtilizationInfo_v1_t

speed

nvmlUnitFanInfo_t

start

nvmlGpuInstancePlacement_t

startTime

nvmlAccountingStats_t

state

nvmlUnitFanInfo_t
nvmlGpuFabricInfo_v2_t
nvmlPSUInfo_t

status

nvmlGridLicenseExpiry_t
nvmlGpuFabricInfo_v2_t
nvmlVgpuLicenseExpiry_t

subClass

nvmlPciInfoExt_v1_t

supportedSchedulers

nvmlVgpuSchedulerCapabilities_t

swRunlistId

nvmlVgpuSchedulerLogEntry_t

T

targetTimeSlice
- nvmlVgpuSchedulerLogEntry_t

time
- nvmlAccountingStats_t

timeRun
- nvmlVgpuSchedulerLogEntry_t

timeRunTotal
- nvmlVgpuSchedulerLogEntry_t

timeslice
- nvmlVgpuSchedulerParams_t
- nvmlVgpuSchedulerSetParams_t

timeStamp
- nvmlProcessUtilizationInfo_v1_t

timestamp
- nvmlFieldValue_t

timeStamp
- nvmlVgpuProcessUtilizationInfo_v1_t
- nvmlVgpuInstanceUtilizationSample_t
- nvmlVgpuInstanceUtilizationInfo_v1_t
- nvmlVgpuProcessUtilizationSample_t
- nvmlSample_t
- nvmlProcessUtilizationSample_t

timestamp
- nvmlVgpuSchedulerLogEntry_t

total
- nvmlMemory_t
- nvmlMemory_v2_t

type
- nvmlBridgeChipInfo_t

U

uiVal
- nvmlValue_t

ullVal
- nvmlValue_t

ulVal
- nvmlValue_t

used
- nvmlMemory_t
- nvmlMemory_v2_t

usedGpuCcProtectedMemory
- nvmlProcessDetail_v1_t

vgpuProcessCount
nvmlVgpuProcessesUtilizationInfo_v1_t
vgpuProcUtilArray
nvmlVgpuProcessesUtilizationInfo_v1_t
vgpuUtilArray
nvmlVgpuInstancesUtilizationInfo_v1_t
vgpuVirtualizationCaps
nvmlVgpuMetadata_t
vgpuVmCompatibility
nvmlVgpuPgpuCompatibility_t
violationTime
nvmlViolationTime_t
vMaxResolution
nvmlFBCSessionInfo_t
volatileCor
nvmlEccSramErrorStatus_v1_t
volatileUncParity
nvmlEccSramErrorStatus_v1_t
volatileUncSecDed
nvmlEccSramErrorStatus_v1_t
voltage
nvmlPSUInfo_t
vResolution
nvmlEncoderSessionInfo_t
nvmlFBCSessionInfo_t

Y
year
nvmlVgpuLicenseExpiry_t
nvmlGridLicenseExpiry_t

Chapter 7. DEPRECATED LIST

Class nvmlEccErrorCounts_t

Different GPU families can have different memory error counters See nvmlDeviceGetMemoryErrorCounter

Global nvmlEccBitType_t

See nvmlMemoryErrorType_t for a more flexible type

Global NVML_SINGLE_BIT_ECC

Mapped to NVML_MEMORY_ERROR_TYPE_CORRECTED

Global NVML_DOUBLE_BIT_ECC

Mapped to NVML_MEMORY_ERROR_TYPE_UNCORRECTED

Global nvmlDeviceGetHandleBySerial

Since more than one GPU can exist on a single board this function is deprecated in favor of nvmlDeviceGetHandleByUUID. For dual GPU boards this function will return NVML_ERROR_INVALID_ARGUMENT.

Global nvmlDeviceGetCurrentClocksThrottleReasons

Use nvmlDeviceGetCurrentClocksEventReasons instead

Global nvmlDeviceGetSupportedClocksThrottleReasons

Use nvmlDeviceGetSupportedClocksEventReasons instead

Global nvmlDeviceGetDetailedEccErrors

This API supports only a fixed set of ECC error locations On different GPU architectures different locations are supported See nvmlDeviceGetMemoryErrorCounter

Global nvmlVgpuInstanceGetLicenseStatus

Use nvmlVgpuInstanceGetLicenseInfo_v2.

Global nvmlClocksThrottleReasonUserDefinedClocks

Renamed to nvmlClocksThrottleReasonApplicationsClocksSetting as the name describes the situation more accurately.

Global nvmlClocksThrottleReasonGpuIdle

Use nvmlClocksEventReasonGpuIdle instead

Global nvmlClocksThrottleReasonApplicationsClocksSetting

Use nvmlClocksEventReasonApplicationsClocksSetting instead

Global nvmlClocksThrottleReasonSyncBoost

Use nvmlClocksEventReasonSyncBoost instead

Global nvmlClocksThrottleReasonSwPowerCap

Use nvmlClocksEventReasonSwPowerCap instead

Global nvmlClocksThrottleReasonSwThermalSlowdown

Use nvmlClocksEventReasonSwThermalSlowdown instead

Global nvmlClocksThrottleReasonDisplayClockSetting

Use nvmlClocksEventReasonDisplayClockSetting instead

Global nvmlClocksThrottleReasonNone

Use nvmlClocksEventReasonNone instead

Global nvmlClocksThrottleReasonAll

Use nvmlClocksEventReasonAll instead